

New Brain

SOFTWARE TECHNICAL MANUAL

Copyright © 1981, 1982, 1983 by Grundy Business Systems Ltd

All rights reserved. No part of this publication may be reproduced, translated, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, magnetic, optical, chemical or otherwise, without prior written permission of the publisher, Grundy Business Systems Ltd, Science Park, Cambridge, UK.

Grundy Business Systems Ltd makes no representations or warranties with respects to the contents hereof and specifically gives no express or implied warranties of suitability or fitness for any purpose. No liability will be accepted for any indirect special or consequential loss or damage arising from the use of this manual. In particular, Grundy Business Systems Ltd reserves the right to make changes in this manual without notification.

NewBrain is a registered trademark of Grundy Business Systems Ltd.

PREFACE

The purpose of this manual is to explain in detail the internal workings and external interfaces of the system software of the unexpanded NewBrain A/AD. This software is implemented mainly in the 8K rom space located at addresses 56K-64K of the NewBrain Z80A processor. Additionally a brief description is given of the NewBrain BASIC dynamic compiler (implemented in the 8K rom space located at 48K-56K) and the NewBrain mathematical package (implemented in 8K system software ROM space located at 56K-64K).

Since the manual was first written a high resolution graphics device driver for the NewBrain has been produced and it is now supplied as standard in NewBrain A/AD models implemented in the 8K rom space located at 40K-48K. Other routines in this space modify the system software and a list of these modifications is given in Appendix C.

Further details of the system software and information relating to modifications are contained in the NewBrain Technical Notes, which should be appended to this manual. The external interfaces of the BASIC, the high-resolution graphics driver and the screen editor are defined in Technical Notes, but details of their internal workings, beyond the information given in this manual, the Handbook and the Technical Notes, is not available to users.

Expanded NewBrain Systems (i.e. those in which the NewBrain A/AD is connected to e.g. an Expansion Interface Module or a Disk Controller Module) make use of additional software including the NewBrain paged memory operating system. Software which makes use of internal features documented here and elsewhere may not be compatible with expanded systems; furthermore the external interfaces of certain software may need to be amended in expanded systems. An outline of the rules to be followed to ensure compatibility is given in Appendix D.

Users of this manual are warned that there may be inaccuracies.

Grundy Business Systems Ltd
Science Park
Milton Road
Cambridge
UK

17 January 1983

<u>CONTENTS</u>	<u>PAGE NUMBER</u>
1. INTRODUCTION	1
1.1 SYSTEM SOFTWARE	1
1.2 USER PROGRAM	5
1.3 DEVICE DRIVERS	6
1.4 HARDWARE	7
1.5 DOCUMENT STRUCTURE	11
2 OPERATING SYSTEM OS	13
2.1 INTRODUCTION	13
2.2 POWER UP	14
2.3 MEMORY MANAGEMENT	19
2.4 KEYBOARD	21
2.5 INTERRUPT SERVICE	23
2.6 INTERCEPT MECHANISMS	33
2.7 SYSTEM QUERIES	35
2.8 DEVICE TABLE	36
2.9 V3 CONTROL	37
2.10 SYNTAX ROUTINES	39
3 IOS	
3.1 INTRODUCTION	43
3.2 OPEN	47
3.3 INPUT/OUTPUT/MOVE	51
3.4 CLOSE	53
3.5 COMMON ROUTINE FOR INPUT, OUTPUT AND CLOSE	57
3.6 FIND DRIVER ENTRY ADDRESS	59
3.7 FIND A MATCHING STREAM NUMBER	61
3.8 FIND MATCHING DRIVER/PORT PAIR	63
3.9 MKBUFF	65
4 CASSETTE DRIVER	
4.1 INTRODUCTION	67
4.2 OPENOUT	71
4.3 OUTPUT	75
4.4 OPENIN	77
4.5 INPUT	79
4.6 CLOSE	80
4.7 WRITE A TAPE BLOCK	81
4.8 READ A TAPE BLOCK	83
5 XIO	
5.1 INTRODUCTION	85
5.2 OPEN	91
5.3 WRITE	97
5.4 READ	101
5.5 CLOSE	105
5.6 RESET	106

CONTENTS continuedPAGE NUMBER

6. XEDIT

6.1 INTRODUCTION	107
6.2 MAIN LINE	108
6.3 EDIT	111
6.4 XEDIT ROUTINES	125

7. KEYBOARD DRIVER

7.1 INTRODUCTION	131
7.2 OPEN	133
7.3 INPUT	134
7.4 OUTPUT	135
7.5 CLOSE	136
7.6 RDKEY	137
7.7 KLOOK	139

8. NEWIO

8.1 INTRODUCTION	149
8.2 LPIO	151
8.3 JGIO	155
8.4 UPIO	157
8.5 DUMMY	158

9. BASIC

9.1 INTRODUCTION	159
9.2 BASIC MAIN LOOP	165
9.3 META MODULES	177

10. MATHS PACK

10.1 INTRODUCTION	181
10.2 ROUTINES	182

APPENDICES

A.1 HARDWARE PORT ALLOCATION
A.2 USER PORT INTERFACE
B LIST OF ROUTINES

CONTENTS continuedPAGE NUMBER

APPENDICES

- C SUMMARY OF CHANGES
- D RULES FOR WRITING PROGRAMS COMPATIBLE WITH
EXPANDED NEWBRAINS
- E NEWBRAIN PROCESSOR MODULES A/AD
Interface, Signal Description and Connector
Pinout
- F NEWBRAIN INPUT/OUTPUT, PORT MAP (this updates
information given in Appendix A)
- G TECHNICAL NOTES

23NOV81-V0-CRSNEWBRAIN-TECH-1.1 01.00

1. INTRODUCTION

This document describes the system software available on all Newbrain processors. There are three current models which offer particular hardware features:

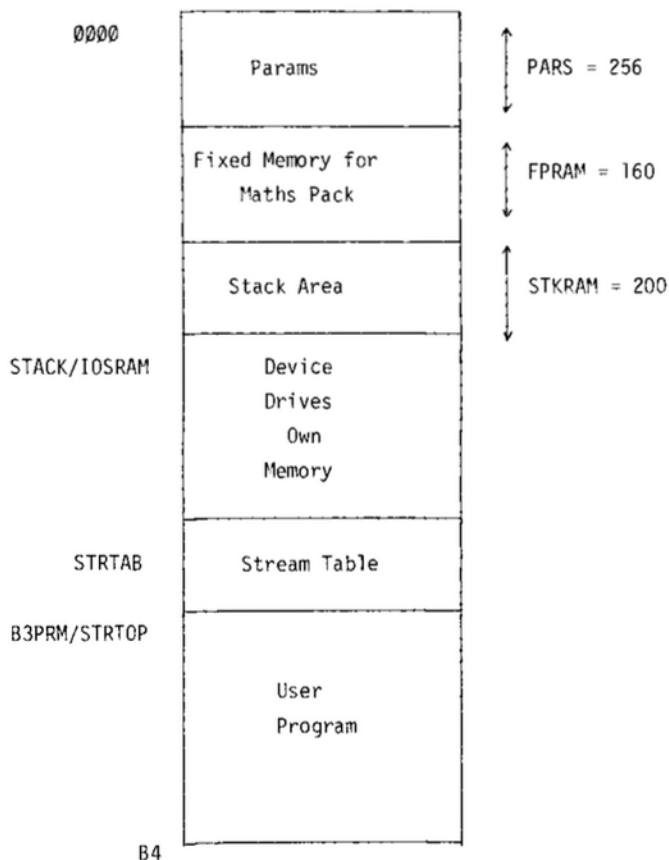
- Model M; 'Page Register', Expansion Port onto Z80 bus, Video, ACIA/CTC, User Port.
- Model A; Expansion Port, Video, no User Port but has software driver serial port - s/w Printer, s/w V24.
- Model V; ACIA/CTC, User Port

Models are suffixed with B and/or D indicating battery back-up and internal VF display respectively.

1.1 SYSTEM SOFTWARE

The system software including BASIC and Floating Point Maths Pack is common to all models though not all models can make use of all the software. The overall control software module is called OS, Operating System, which gains control at power up and performs initial memory validation and initialisation. An own memory area is maintained in fixed memory locations 0-FFH. Interrupt handling and other top level functions are performed by OS. These are provided by routines internal to OS, the entry points of which are available through a routine address table.

OS and IOS may be considered as permanently active tasks. Both support own memory areas constantly in use through User program request or directly by the User program. OS supports the page zero (first 256 bytes) parameters and IOS maintains a table of active devices, the Stream Table, and a device driver own memory area. The Random Access Memory organisation determined by OS and IOS is shown below:



23NOV81-VO-CRS

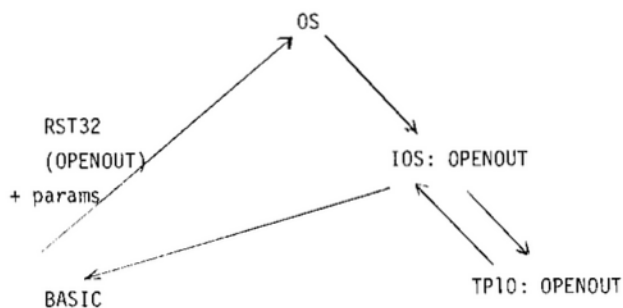
NEWBRAIN-TECH-1.1

03.00

The address STACK, the base of the stack, and IOSRAM which is the start address of Device Driver Own Memory are concurrent. The Stack Pointer is initialised by OS but may be changed to point into the User Program memory if a larger stack area is required. B3PRM marks the base of the User Program and is normally concurrent with the top of the Stream Table. When IOS requires memory, at the time of a new Stream Table entry, then the User Program is requested by OS to provide the desired memory space. If the User Program can supply this memory B3PRM is temporarily set on the new base of the User Program area. When the table entry is subsequently made STRTOP becomes equal to B3PRM again.

The Operating System links to the Input/Output System, IOS by means of an intercept mechanism in OS. User programs which require IO operations make an appropriate call through OS which translates this to an IOS request through a routine address table. IOS is itself a level of software above both the user program making the I/O request and the device driver modules actioning the request. OS functions are available through the same mechanism.

The implementation of a typical User Program I/O request is shown here graphically.



23NOV81-V0-CRS

NEWBRAIN-TECH-1.1

04.00

BASIC, as an example of a User Program wants to open the cassette driver in preparation for a data output to tape. BASIC makes the request through the intercept routine in OS, passing it the IOS routine request with parameters in the registers. The IOS routine is called and from the passed parameters determines the driver and its open routine. On completion of the open the driver passes control back to IOS which returns to the User Program. It is possible to call IOS, the driver routine, or even directly drive an I/O device from a User Program, but whether the user is implementing a new driver or his own User Program it is advisable, and indeed good software practice, to adopt the Newbrain standard interfaces.

1.2 USER PROGRAM

The User Program becomes active when it is called by OS after initialisation. Its memory allocation is determined by the values of B3PRM which marks the base address and B4 which marks top of available RAM. It is expected that User Programs are designed such that all memory accessing is relative and that the memory boundary addresses need not be referred to after User Program initialisation. To this end, the IX and IY registers are maintained by all system software routines. On entry no devices are open but default devices have been set up by OS in the page zero params though these may be ignored. Three program modules are present in a User Program:-

- The User Program itself providing the desired application function.
- A Make Space module which performs the processing necessary when OS makes a memory request from the User Program.
- A Return Space module which performs the processing necessary when OS returns memory to the User Program expanding the User Program space.

The address of the memory space modules are set by the User Program into the page zero parameters. Although the system supplied User Program BASIC dynamically adjusts its internal base address, a fixed application User Program, the maximum IOS requirements of which can be determined, can reserve sufficient space at its low memory addresses for any OS memory requests. In this case the Make Space and Return Space routines become trivial.

23NOV81-V0-CRS

NEWBRAIN-TECH-1.3

01.00

1.3 DEVICE DRIVERS

Supported through IOS are a number of code paths which perform I/O operations for the following devices:-

- Twin cassette tape driver
- TV and VF display drivers
- Keyboard drivers
- Software timed Line Printer driver
- Software timed V24 driver.

The device driver routines are seldom active and only by User Program request. Typically, while any of these are processing, other system operations are curtailed. Most notably the video display is disabled during data transfer to avoid data loss and to maintain synchronism for software timed serial data I/O.

Two display types are supported which may be driven individually or simultaneously. The Video structure supports 40 and 80 character per line and from 1 to 30 lines display lines. Various hardware options are supported. The VF display is a single line 16 character unit mounted within the Newbrain processor. By default the internal software line length is 80 characters, the full line contents being viewed by scrolling left or right. The device driver which interfaces to the display hardware is supported by a comprehensive display editor package.

1.4 HARDWARE

The Newbrain processor is based on the Z80 device supported by an intelligent IO microprocessor, the COP 420 GUW. Via a number of protocols the COP communicates with the Z80 passing data and control information in order to drive the following:-

- cassette
- keyboard
- VF display

The COP maintains a status check on battery condition and whether the STOP keyboard key has been depressed.

On battery models of the Newbrain the Z80 may power down while waiting for a keyboard key to be depressed and can command the COP to power down at this time. Control circuitry and ROM modules associated with these devices are powered down with them. Four independent power circuits supply hardware elements within the Newbrain:-

- V0 power is always present and supplies the random access memory and its control.
- V1 is the COP and related circuitry supply.
- V2 is the Z80 supply.
- V3 supplies power hungry IO devices such as video.

The Newbrain only supports the Maskable Interrupt Mode 1, non-maskable interrupts are disabled by holding the NMI pin high.

Priority servicing is determined by software with the COP having the highest priority in the OS interrupt handler. The COP service routine does not enable interrupts and runs for a maximum of 600 microseconds. Interrupts are disabled during the serial transfer of a byte in the software timed drivers. The duration of the disable is dependent on baud rate. Mode 1 interrupts are vectored through location 38H equivalent to RST56. The other Restart locations are used as follows:-

Restart	Use
RST8	May be used by any routine which should redefine it if control passes to other users. Currently these are the tape driver and display driver.
RST16	} May be used by the User Program for any purpose. Used by BASIC as intercept locations through OS.
RST24	
RST32	} Intercept locations for OS, Maths Pack and IOS.
RST40	
RST48	Reserved for use within interrupt handling software. Note RST8 should not be used within interrupt handler.

The Z80 performs IO through hardware ports. Thirty-two ports are currently defined, port numbers of 32 and greater are reserved for future expansion. Two ports 7 and 20, the Enable and Status registers respectively are important in controlling the Newbrain operation and are frequently driven by operating software. Two formats of Enable register are defined:-

- Model A has interface to software driven V24 devices
- Other models with hardware driven V24.

Bits 7 and 6 of the Enable Register are also used to define one of four possible conditions of the Status register bit 0. Similarly bits 5 and 4 of the Enable Register select a signal for bit 1 of the Status Register. No conflict is generated by the dual use of bits 7 to 4 of the Enable register as they are validated in their V24 mode by signals at other ports. An OUT instruction to the Enable Register with the appropriate bits set results in the selection of data for bits 0 and 1 of the Status Register on subsequent IN instruction. The structure of these registers are shown overleaf in Table 1.4. A full description of all ports is given in Appendix A.

Model A Enable Register

Bit 0 = <u>C</u> lock Enable	Bit 1 = Not Used
Bit 2 = TV Enable	Bit 3 = Not Used
Bit 4 = <u>C</u> lear for Sending	Bit 5 = Transmit data V24
Bit 6 = Not Used	Bit 7 = Transmit data Printer

Not Model A Enable Register

Bit 0 = <u>C</u> lock Enable	Bit 1 = <u>U</u> ser Enable
Bit 2 = TV Enable	Bit 3 = V24 Enable
Bit 4 = V24 Sel.Rx.Bit 0	Bit 5 = V24 Sel.Rx. Bit 1
Bit 6 = V24 Sel.Tx.Bit 0	Bit 7 = V24 Sel.Tx. Bit 1

V24

Selection Rx 00 = Tape Loop 0, 01 = Tape Loop 1
 10 = Not Used, 11 = Serial Comms.
 Tx 00 = Tape Loop 0, 01 = Tape Loop 1
 10 = Printer 11 = Serial Comms.

Status Register

	11	10	01	00	Enable Reg Bits
Bit 0 = <u>C</u> all Ind, tape in, 40/80, Excess					7, 6
Bit 1 = <u>C</u> all Ind, Bee, TV console, Power Up					5, 4
Bit 2 = Mains Present				Bit 3 = <u>U</u> ser Status	
Bit 4 = <u>U</u> ser Interrupt				Bit 5 = <u>C</u> lock Interrupt	
Bit 6 = <u>A</u> CIA Interrupt				Bit 7 = <u>C</u> OP Interrupt	

Table 1.4

1.5 DOCUMENT STRUCTURE

The other sections within the Newbrain Software Technical Description describe the Operating System, IOS and Standard Device Drivers, and gives a structural overview of BASIC and the Maths Pack. Certain routines within the operating software are particularly useful and the interface conditions to these are given in the following format:-

Entry:	Specifies the entry point of the routine and identifies those registers which have specific contents used by the routine.
Function:	A brief description of the operation performed by the routine.
Routines:	Where appropriate any major subroutines called are named with section numbers where they are described.
Exit:	The register contents on exit from the routine are described. Carry is clear for a normal exit, set to 1 if an error condition has been detected. IX and IY are always maintained and are not mentioned in the preserved register list.
Params:	Lists the page zero variables used by the routine. An asterisk alongside the routine indicates that the variable may be updated.
Ports:	Any ports written to or read by the routine are listed.

23OCT81-VO-CRS

NEWBRAIN-TECH-2.1 01.00

2.1 INTRODUCTION

The Newbrain Operating System, OS, is a collection of routines, some of specific functions and used internally and some more general and called by external programs. The power up and interrupt service routines are an example of the former and since the operation of reading a keyboard key is closely associated with the hardware and power-down this routine is also present.

OS also contains fundamental routines called from external programs. The intercept mechanisms allow users to call routines through the operating system via tables of addresses enabling specific calls to be re-routed if necessary. The Operating System is also the repository of other simple routines of general interest - syntax, memory management, power control, system queries and is the location of the standard device table.

2.2 POWER UP

As power is applied to the processor the E000 ROM is decoded at each 8k page in the Z80 address space. Thus the instruction at location 0000 at this time is identical to that at E000 i.e. JP E003. This has the effect of loading the PC with E003 from which address the next instruction is taken. This instruction sets the interrupt mode to 1. The Enable Register port is written to with TV disabled and selecting EXCESS and PWRUP, bits 0 and 1 of the Status Register. The power up sequence loops until the PWRUP bit becomes set.

2.2.1 Memory Test

The power up procedure then enters the memory test sequence. The RAM test performs four write/read check loops writing four bit-patterns to each location and complementary bit patterns to adjacent locations, i.e. on first pass all zeroes and all ones are written to adjacent locations throughout RAM; on second pass 55H and AAH are written to adjacent locations etc. On the read check phase the contents of each location are compared with the written value. Each location is checked until either top of RAM, hard coded as 9FFF, is reached or an error is detected. If an error then the top of RAM is set as the immediately previous memory address.

At the end of the memory test the register pair DE contains either the address of top of RAM + 1, or if a memory error detected the last "good" odd (not divisible by two) memory address + 1.

2.2.2 System Parameters

Following the memory test the default system parameters hard coded in the E000 ROM are block moved down to the 0000 RAM. A list of parameters is given in Table 2.2 The following default parameters are then immediately modified/verified:

- 1) B4, set to the highest "good" RAM address + 1.
- 2) If the EXCESS bit of the status register is set, the EXCESS parameter is modified from its default value of 4 to 24. EXCESS is used by the video device and described in section 5.
- 3) If the Mains Present bit is set and the TVCNSL and 40/80 bits of the Status Register are zeroes, then the default parameter string length (DPSL) of 3 is not changed specifying an 80 character 24 line video device. Otherwise DPSL is set to 0, which if the console is a video specifies a 40 character, 24 line display.
- 4) The default console output device (DEV0) number is set to 0 which defines a video display as the console device. If the Mains Present bit is 0 or the TVCNSL bit is reset then the DEV0 parameter is set to 3 specifying the VF display as the console output device.

Initialisation complete, the start up sequence ends by testing whether an expansion ROM is fitted at location A000. If so, its first byte is 00H. If Bit 7 of the contents at A000 is zero, then OS jumps to A001. Otherwise, interrupts are enabled, the user address is retrieved from system parameter -UP and jumped into.

230CT81-V0-CRS

NEWBRAIN-TECH-2.2 03.00

<u>Address</u>	<u>Length</u>	<u>Name</u>	<u>Value</u>
00	1	DI	Disable Interrupt instruction.
01	3	TPON	Transfer for RST0, for wake-up after Z80 power down
04	2	B3PRM	Lower bound of RAM allocation to user program initially set concurrent with STACK top
06	2	B4	Upper bound + 1 of RAM allocated to user program
08	3	TRST8	Transfer for RST 8, used by maths and cassette dd
0B	1	DEV0	Device driver number of default console, used by BASIC
0C	1	EXCESS	For TV device driver defaults to 4
0D	1	TV0	For TV device driver - flash clock initially set to zero
0E	1	TV2	For TV device driver - cursor flags initially set to zero
0F	1	TV1	For TV device driver - character at cursor initially set to zero
10	3	TEXM	Transfer for RST16, user program intercept "call by name"
13	1	DEV2	Number of BASIC default store device, normally cassette
14	2	SAVE2	IX save by IOS(for communication between IOS &
16	2	SAVE3	IY save by IOS(user program)
18	3	TEXMNC	Transfer for RST24, user program intercept, "call no carry by name"
1B	1	PLLEN	Print line length for TAB and POS in BASIC
1C	1	PHPOS	Print head position for TAB and POS i.
			BASIC

Table 2.2 System Parameters

230CT81-V0-CRS

NEWBRAIN-TECH-2.2 04.00

<u>Address</u>	<u>Length</u>	<u>Name</u>	<u>Value</u>
1D	1	PZLEN	Print zone length for TAB and POS in BASIC
1E	2	SAVE1	SP save for power down
20	3	TRST32	Reserved for RST32 use by OS intercept, "call by name"
23	1	BRKBUF	Used by BRKTST and other routines which detect break-in conditions (e.g. cassette driver)
24	1	ENREGMAP	Last byte sent to the enable port register
25	1	IOPUC	I/O power user count initially set to zero
26	2	UP	User program address defaults to BASIC
28	3	TRST40	Reserved for RST40 use by OS intercept, "call no carry by name"
2B	1	KBMODE	Graphics and shift key lock status initially 0.
2C	2	GSPR	User program make space routine address
2E	2	RSPR	User Program return space routine address
30	3	TRST48	Reserved for RST48
33	1	BUFLG	When set indicates buffers in motion, initially 0.
34	2	DPSP	Default parameter string pointer for device 0, initially points to string "L24"
36	2	DPSL	Default parameter string length for device 0, initially set to 3
38	3	TINT	Transfer for RST56(mode 1 device interrupt service)
3B	1	COPCTL	COP control, interrupt acknowledge byte written to COP in response to a COP interrupt
3C	1	COPST	COP status, set by COP interrupt service routine

Table 2.2 System Parameters
cont.

230CT81-V0-CRS

NEWBRAIN-TECH-2.2 05.00

<u>Address</u>	<u>Length</u>	<u>Name</u>	<u>Value</u>
3D	1	COPBUFF	Data characters to COP
3E	16	OUTBUFF	VF display buffer initially set to spaces
40	2	TIMBUFF	COP internal counter value used to time the 1 minute power down interval
50	2	CHKSUM	Tape transfer accumulated checksum, initially 0.
52	4	CLACK/CLOCK	Count of COP interrupts
56	2	STRTAB	Address of start of stream table initially set concurrent with STACK top
58	2	DEVTAB	Address of start of device table initially points to hard coded table in E000 ROM
5A	2	TVCUR	Address of cursor position
5C	2	TVRAM	Start address of TV own memory
5E	2	OTHER1	Reserved for use by driver which needs to maintain page zero memory address variable if own memory moves.
60	2	OTHER2	
62	2	IOSRAM	Address of device driver own memory area for streams initially set concurrent with STACK top.
64	2	STRTOP	Address of end of stream initially set concurrent with STACK top
66	3	TNMI	Transfer for NMI (not used, NMI pin strapped high)
69	3	FICLKM	Frame interrupt counter, while video device is active
6C	1	TBRP	Transmit baud rate parameter, default setting is 2
6D	1	RBRP	Receive baud rate parameter, default setting is 2.

Table 2.2 System Parameters cont.

23OCT81-V0-CRS

NEWBRAIN-TECH-2.3 01.00

2.3 MEMORY MANAGEMENT

Entry: SPACE, GSPCBK
BC = number of bytes

Function: Provides an interface with the user program
get space and return space routines.

Routines: SPACE
GSPCBK

Exit: Carry set if error detected (only for SPACE)
A = error code 100 = no memory space available
BCDE preserved

Params: GSPR
RSPR
B3PRM *
SAVE2 *
SAVE3 *

On entry to these routines BC contains the number of bytes required or to be returned. The user program routines are held in page zero locations GSPR and RSPR, get space and return space respectively. The user program index registers are saved in page zero location SAVE2 and SAVE3 and are restored prior to the CALL into the user space routine.

On return the user program index registers are re-saved in SAVE2 and SAVE3. The page zero location B3PRM, which contains the memory address which is the lower bound of user space, is updated either by adding BC to it in the event of a SPACE call, or subtracting BC bytes from it in the case of GSPCBK.

230CT81-V0-CRS

NEWBRAIN-TECH-2.3

02.00

The user program is supplied the number of bytes required or returned in BC. IX and IY are set on entry to the user program space routines as they were on previous exit from the user program. I.e. OS maintains the user program index registers.

On return from the user program space routine BC DE are expected to have been preserved. The condition of IX and IY replaces the previously maintained user program index registers. I.e. next time the user program regains control IX and IY are as modified by the user space routines.

230CT81-V0-CRSNEWBRAIN-TECH-2.4 01.00

2.4 KEYBOARD

Entry: IMMKEY, GETKEY

Function: Gets a byte of keyboard data from COP; IMMKEY returns immediately with or without data; GETKEY waits until a key depressed.

Routines: IMMKEY
GETKEY

Exit: A = keyboard data byte, carry clear if valid, carry set if no data available, GETKEY never sets carry. All registers including alternates may be destroyed since Z80 can power down while waiting for a key.
IX and IY are still preserved.

Params: COPST *
COPCTL *
TIMBUFF*

The GETKEY routine does not return to caller until a keyboard key has been depressed and a byte passed via the COP to the Z80. IMMKEY returns immediately to caller picking up a data byte if one is available.

2.4.1 IMMKEY

Interrupts are disabled and the KDATA Bit 6 of page zero location COPST (COP status) is tested and always reset. The contents of COPBUFF are written to the accumulator. Interrupts are then enabled.

23OCT81-V0-CRS

NEWBRAIN-TECH-2.4 02.00

If KDATA was set on entry indicating that COPBUFF contained a data byte then IMMKEY returns to caller with carry clear and accumulator containing data byte. Otherwise returns with carry set. BCDE are preserved.

2.4.2 GETKEY

The GETKEY routine only returns to caller once the COP has written a byte to COPBUFF and set KDATA, Bit 6 of page zero location COPST (COP Status). The IMMKEY routine is used to clear the COP interface.

The index registers are saved and IMMKEY called. This has the effect of clearing the KDATA bit ~ no data is associated with this call. The GETKEY routine then enters a wait loop. Interrupts are disabled and the KDATA bit tested. If not set then the last written byte to the Enable Register is examined and if the IOPOWER Bit 3 is set, interrupts are enabled and the wait loop re-entered. If the IOPOWER Bit is zero the Z80 enters a power down routine. If the READY Bit 5 of COPST is not set or the TIMEOUT Bit 4 of COPST is set then interrupts are enabled and a HALT is executed powering down the Z80. Otherwise the command B8 (updates timer, disables regular interrupts) is written to COPCTL and TIMBUFF is loaded with the 1 minute power down count prior to the EI and HALT instructions.

Eventually a keyboard key will be depressed resulting in a power up or normal interrupt. The routine is forced back into the wait loop where KDATA is tested and should be true. Having reset the TIMEOUT bit of COPST, the IMMKEY routine is performed prior to return to caller. The accumulator will contain the data byte and carry will be clear. AFBCDEHL and alternate registers are destroyed if Z80 powers down.

230CT81-V0-CRSNEWBRAIN-TECH-2.5 01.00

2.5 INTERRUPT SERVICE

Entry: INTRPT

Function: Handles COP and Frame Interrupts

Routines: FRINT section 2.5.2
COPIS section 2.5.1

Exit: AFBCDEHL preserved

Two interrupt service routines are currently supported:-

- Frame (Clock) Interrupt
- COP Interrupt

Interrupts are defined on power-up to be mode 1 which force a restart through location 38H. This results in a jump to the location INTRPT. The Status Register is read immediately as some devices drop their interrupt lines quickly. AFBCDEHL are saved on the stack. The interrupt bits of the Status Register are tested in the sequence COP, Clock. If an interrupt bit is set the specific interrupt handling routine is entered.

On return, AFBCDEHL are popped off the stack, interrupts are enabled and normal program execution continues.

230CT81-V0-CRS

NEWBRAIN-TECH-2.5 02.00

2.5.1 COP Interrupt Handling

Entry: COPIS

Function: Handles COP interrupts

Routines: REGINT section 2.5.1.1
 CASSERR section 2.5.1.2
 CASSIN section 2.5.1.3
 KBD section 2.5.1.4
 CASSOUT section 2.5.1.5

Exit: Normally returns through one of routines branched to.
 Abnormally on unrecognised COP interrupt returns to caller

Params: COPCTL

Ports: COP

The COP I/O port is read. The contents of COPCTL (COP control), set previously by a routine requiring a specific COP service, are written to the COP I/O port both as an acknowledgement of the COP interrupt and as the issue of the next COP command. The top 4 bits of the COP input byte determines the type of interrupt.

Value of Bits 4-7

Interrupt Type

0	REGINT, Regular Interrupt occurs at approximately once every 12.5 mS
1	CASSERR, Cassette error
2	CASSIN, Cassette input
3	KBD, Keyboard
4	CASSOUT, Cassette Output

23OCT81-V0-CRSNEWBRAIN-TECH-2.5 03.00

Each of these COP interrupt types has its own service routine. Any other value of bits 4-7 results in an immediate return out of the COP interrupt handler.

2.5.1.1 REGINT

Entry: REGINT

A = COP input byte

C = COP port number

D = COP acknowledgement/command byte

HL points at COPST

Function: The operations to support the previous issued command byte are performed plus general Regular Interrupt housekeeping

Exit: No special conditions, normal return, though abnormally can jump to power off instruction

Params: COPST *

COPCTL *

CLACK *

Ports: Status Register

COP

The bits 0-2 of the COP input byte are or'd into the corresponding bits of COPST (COP Status). The READY bit 5 of COPST is set. By default the next COP command is set to NULLCOM(D0H) and this is written to COPCTL. Several of the output commands sent to the COP on entry to the COP Interrupt Handler, 2.5.1, require additional operations.

If the command was DISPCOM or TIMCOM, then a synchronous data transfer to the COP is performed. Synchronisation is achieved when both COP and Clock interrupt states bits occur simultaneously (though a Clock interrupt need not occur).

23OCT81-VO-CRS

NEWBRAIN-TECH-2.5 04.00

At this time the data transfer begins, at a 1 byte per 20 microsecond rate. For DISPCOM 18 bytes are transferred. The first 2 bytes are "don't care", the remaining 16 are the contents of OUTBUFF. TIMCOM is supported by the transfer of 6 bytes. Only the middle 2 bytes are meaningful and are the contents of TIMBUFF.

If the previous command was NULLCOM then the routine determines whether a full power down can be initiated. If both the TIMER0 Bit 0 of COPST (determined by COP input byte) and the TIMEOUT Bit 4 of COPST are set then the PDNCOM is written to the COP port. The timeout bits of COPST are cleared and DISPCOM is set in COPCTL so that on power up the VF display is refreshed from OUTBUFF. The routine then jumps through TOS to the POFF (Power Off) location which executes a HALT.

Except when a full power down is initiated the REGINT routine ends by incrementing the Regular Interrupt count held in quadruple precision in CLACK.

2.5.1.2 CASSERR

Entry: CASSERR
Registers as for REGINT

Function: Sets READY and CERR Bits 5 and 4 of COPST and sets Error Code 130, Tape Drop Out, in COPBUFF

Exit: No special conditions

Params: COPST *
COPBUFF*

23OCT81-V0-CRS

NEWBRAIN-TECH-2.5 05.00

2.5.1.3 CASSIN

Entry: CASSIN
Registers as for REGINT

Function: Reads a byte of data from the COP, if one is present;
else tests whether the stop key has been depressed.

Exit: No Special Conditions

Params: COPST *
COPBUFF *
BRKBUF

Ports: Enable Register
COP

The 1sb of the COP input byte is the Run-in bit indicating, when set, that no data is associated with this interrupt. If bit 0 of COP input byte is set, then bit 1 is tested and if set indicates the break key has been depressed. If break-in is allowed i.e. BRKBUF is zero then BRKOK, READY and CERR Bits 3, 5 and 4 of COPST are set and COPBUFF is set to zero.

If bit 0 of the COP input byte is zero then the COP will set a data byte at the COP port 290 microseconds after the COP interrupt which will remain until 530 microseconds after the COP interrupt. A software countdown is executed and the COP port read. The retrieved data is set in COPBUFF. Meanwhile the Enable Register is written to resetting TVENA so that the video does not make a bus request during the COP data transfer with the possibility of losing the data.

230CT81-V0-CRS

NEWBRAIN-TECH-2.5 06.00

2.5.1.4 KBD

Entry: KBD
Registers as for REGINT

Function: Reads a data byte from the COP

Exit: No Special conditions

Params: COPST *
COPCTL
COPBUFF*

Ports: COP

For a Keyboard Interrupt a data byte becomes valid at the COP port 290 microseconds after the COP interrupt. KBD detects the data byte, not by timing 290 microseconds, since video is not disabled, but writing a byte to the COP port and waiting for it to change.

The data byte output by the COP is the coordinates of the depressed key of the keyboard. KBD writes 80H to the COP port, which is not a valid coordinate, and enters a wait loop which includes a read of the COP port. When the data read is not equal to 80H, the wait loop is exited, the port is read again in case the data was in transition previously. The KDATA Bit 6 of COPST is set and the COP input data byte is written to COPBUFF. If the BRKOK Bit 3 of COPST is set, then if the data byte refers to the "*" key, both KDATA and BRKOK are reset prior to return. For hardware reasons on certain machines the "*" key performs the stop key function during cassette searching.

230CT81-V0-CRSNEWBRAIN-TECH-2.5 07.00

2.5.1.5 CASSOUT

Entry: CASSOUT
Registers as for REGINT

Function: Writes a byte of data to the COP

Exit: No Special Conditions

Params: COPST *
COPBUFF

Ports: COP
Enable Register

TVENA is reset by writing to the Enable Register. READY is set in COPST. The data byte held in COPBUFF must be written to the COP between 290 and 530 microseconds of the COP interrupt. The interval is timed by a software countdown and the byte written.

2.5.2 Clock Interrupt Handling

Entry: FRINT

Function: Handles Clock interrupts

Exit: Normal return to caller

Params: FICKM *
BUFLG
TV0 *
TV1
TV2 *
TVCUR

Ports: Clear Clock Interrupt, port 4
Load TV Addresses, Ports 8 and 9
Enable Register

The Clock Interrupt is cleared by executing an OUT instruction to port 4. The Frame Interrupt counter, FICKM, in Page Zero is incremented. This is a triple precision count which, provided clock interrupts are enabled, gives an accurate 20ms clock count.

BUFLG is tested. This is set by IOS during execution of a CLOSE when stream tables and own memory are being moved, see section 3. If BUFLG is set then the TV Enable Bit 2 of the Enable Register is reset effectively blanking the video. Normally BUFLG is zero so interrupt handler continues with cursor handling. The active locations in page zero are:-

TV0, contains the cursor flash clock

TV1, contains the display character at the cursor position

TV2, contains cursor status bits

Bit	Value
0-3	Unused
4	1 = Blank Video (For this frame only)
5	0 = Cursor is blob character 1 = Cursor is underline
6	1 = Cursor displayed 0 = Cursor not displayed
7	0 = Blank Video

TVCUR, 2 bytes, containing the RAM address within the TV store
which is the cursor position address

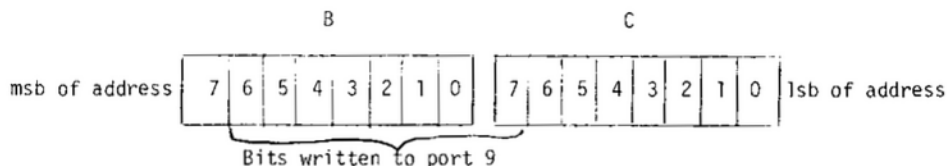
The flash clock, TV0, is incremented. Bits 4 and 7 are tested and if necessary TV Enable is reset. Bit 4 of TV2 is always reset by this routine.

If a cursor is to be displayed, Bit 6 of TV2 set, then the flash clock in TV0 is interrogated. Each 16 counts of TV0 causes the cursor to be alternately displayed and not displayed. Since the Clock Interrupt occurs once every 20mS this gives a cursor flash cycle time of 640mS. If Bit 4 of TV0 is set then the cursor, blob or underline (determined by Bit 5 of TV2), is set at the cursor address held in TVCUR. (Cunningly if the character at the cursor position is the cursor character, then during the cursor display phase of the flash cycle a space is written at the TVCUR address). If Bit 4 of TV0 is zero then the character at cursor position held in TV1 is written to the TVCUR address.

Interrupts are enabled and the SETFRM routine, see section 6, is called. This places the start address of TV RAM store into BC. This address can be considered as:

23OCT81-V0-CRS

NEWBRAIN-TECH-2.5 10.00



The TV store start address always occurs on a 64 byte boundary and in the address range 0-32k and can therefore be specified in 9 bits. The most significant bits of the address are written to port 9. If Bit 6 of C is set then an OUT to port 8 is executed. The OUT operation to port 8 causes the Bit 6 of the TV RAM address to be set. For output devices with an excess of 24, the Video Bit 2 of the Enable Register is set, the Enable Register written and the routine exits. Other devices expect a further OUT instruction to be executed. This does not write to a specific port but outputs the full 16 bit address of the start of TVRAM store onto the address bus. Video is then enabled as before.

23OCT81-V0-CRSNEWBRAIN-TECH-2.6 01.00

2.6 INTERCEPT MECHANISMS

Entry: via RST16, RST24, RST32, RST40
TOS points to an entry in ZOSTABLE or ZTABLE

Function: Routes calls via a table of routine addresses.

Routines: CALLOS
COSNC
CALLRTN
CALLNC

Exit: Jumps into required routine address
TOS points to originator of RSTnn return address
AFBCDEHL preserved.

The operating system provides a simple table based calling mechanism which makes use of the RST instructions of the Z80. Forcing subroutine calls through the standard intercept code gives a number of benefits:-

- solves the global symbol problem between ROMS
- allows the easy addition of routines
- maintenance

Two tables of global symbols support the intercept code:-

- ZTABLE contains the user program (BASIC) routine addresses
- ZOSTABLE contains operating system and maths pack routine addresses

The tables define two byte entries of the form:-

RSTnn	
Offset	;word offset of routine address in ZTABLE/ZOSTABLE

which are linked into the callers code. The operating system uses RST32 and RST40 which cause a jump to CALLOS and COSNC respectively. The user program uses RST16 and RST24 which cause a jump to CALLRTN and CALLNC respectively. Both sets of restarts behave identically except for operating through their own tables.

2.6.1 CALLOS

The registers AFBCDEHL are exchanged with the alternate register set. After the execution of the RST32 the 'return address' pushed onto the stack is actually the address of the offset to the routine address in ZOSTABLE. The real return address is set on the stack by incrementing the address at the stack pointer. The byte address of the routine address in ZOSTABLE is calculated to be

$ZOSTABLE + 2 * Offset$ where ZOSTABLE is the base address of the table.

The contents at this address is the start address of the required routine. The routine start address is pushed onto the stack, the registers are restored and a RET instruction is performed causing a jump into the routine.

2.6.2 COSNC

This executes as CALLOS but only if carry is not set. If carry is set the correct return address is set on the stack and the routine returns to caller.

23OCT81-VO-CRS

NEWBRAIN-TECH-2.7 01.00

2.7 SYSTEM QUERIES

There are two system query routines currently implemented. BRKTST tests and clears the break key flag; STKTST checks how much space is left on the stack.

2.7.1 BRKTST

Entry: BRKTST

Function: Tests and clears the break key flag

Exit: If flag set A=0 carry is set
Else carry clear
BCDEHL preserved

Params: BRKBUF
COPST *

The Page Zero location BRKBUF is examined. If non-zero then routine makes immediate return to caller with carry clear. If BRKBUF is zero then the BRKKEY Bit 2 of COPST (COP Status) is tested and reset. If set then A is set to zero and carry is set. If BRKKEY is zero then A contains BRKBUF contents and carry is clear. Preserves BCDEHL.

2.7.2 STKTST

Entry: STKTST
BC = number of bytes

Function: Determines whether BC bytes will fit in the remaining stack space.

Exit: carry set if insufficient stack space BCDE preserved

230CT81-V0-CRS

NEWBRAIN-TECH-2.8 01.00

2.8 DEVICE TABLE

The address of the device table is found in the Page Zero location DEVTAB. The offset zero of the device table is the count of number of entries in the table. The devices supported are

<u>Offset</u>	<u>Device</u>
0	11 = number of device table entries
1	TVIO Video console output
3	TP1IO First cassette unit
5	TP2IO Second " "
7	LIIO VF display
9	TLIO Combined Video and VF display
11	KBWIO Keyboard wait for a key
13	KB1IO Keyboard immediate return with a key if available
15	UPIO User port
17	LPIO Line printer
19	JGIO Software driven V24
21	DUMMY Does nothing, may be used for directing discardable output

Further details of device drivers may be found in section 3.

23OCT81-VO-CRS

NEWBRAIN-TECH-2.9 01.00

2.9 V3 CONTROL

Each time a device is opened which uses V3 Power the routine IOPON is called. Similarly each time a device is closed which uses V3 Power the routine IOPOFF is called. Both routines maintain a count in IOPUC which is incremented each time IOPON is called and decremented each time IOPOFF is called.

2.9.1 IOPON

Entry: IOPON

Function: Increments IO Power Use Count and conditionally
enables IO Power

Exit: No special conditions
BCDEHL preserved

Params: ENREGMAP *

IOPUC *

Ports: Enable Register

Interrupts are disabled. The last byte written to the Enable Register is held at ENREGMAP. The IOPOWER Bit 3 of this byte is set and the new contents of ENREGMAP written to the Enable Register. Interrupts are then enabled. The IOPUC (IO Power Use Count) is incremented. If prior to being incremented the IOPUC was zero then ACINIT is called which initialises the ACIA and CTC.

230CT81-V0-CRSNEWBRAIN-TECH-2.9 02.00

2.9.2 IOPOFF

Entry: IOPOFF

Function: Decrements IO Power Use Count and conditionally
disables IO PowerExit: No special conditions
BCDE and Carry preserved.Params: ENREGMAP *
IOPUC *

Ports: Enable Register

IOPUC is decremented. If as a result IOPUC becomes zero then the IOPower bit of ENREGMAP is reset and the new contents of ENREGMAP written to the Enable Register switching off V3 Power.

230CT81-VQ-CRS

NEWBRAIN-TECH-2.10 01.00

2.10 SYNTAX

Four useful routines are available for character and numeric string manipulation:-

- lower to upper case conversion
- numeric character to binary
- numeric string to 16 bit binary
- numeric string to 8 bit binary

2.10.1 Lower to Upper Case Conversion

Entry: HLTTCAPS

HL contains address of character for conversion

or

TTCAPS

A = character for conversion

Function: If the character is one of the twenty-six lower case letters then it is converted to its upper case equivalent.

Exit: A = converted character

Carry is set if the input character has an ASCII value less than 61H

BCDEHL preserved

230CT81-V0-CRS

NEWBRAIN-TECH-2.10 02.00

2.10.2 Convert ASCII numeric character to binary

Entry: RDIGIT

A = numeric character

Function: If the character in the accumulator is one of the ten numeric ASCII characters, it is converted to its binary equivalent.

Exit: A = binary number, Carry is clear

Else A = input character, Carry is set indicating that the input character was not an ASCII numeric.

BCDEHL preserved

2.10.3 Convert ASCII numeric character string to 16 bit integer

Entry: RDINT

C = a string character count

HL points to first numeric character held in memory; the character in the lowest memory address is the most significant digit.

Function: Converts an ASCII numeric character string or sub-string to a sixteen bit integer.

Exit: C = count on entry less number of numeric characters read.

DE= binary integer

HL contains address following the address of last numeric character read.

Carry clear if no errors

Carry set if overflow or first character read was non numeric

B preserved.

230CT81-V0-CRS

NEWBRAIN-TECH-2.10 03.00

This routine is specifically designed for converting numeric ASCII sub-strings from within I/O parameter strings. The entry parameter C is the total count of characters yet to be processed in the string and not the number of numeric characters for conversion. This parameter is maintained by the routine, decremented for each character processed, but if it becomes zero the routine is exited.

The routine processes character by character until a non-numeric ASCII value is detected. If this is the first character then an error is returned - carry is set. To ensure successful conversion it is imperative that the numeric string has a trailing delimiter so that the routine stops converting.

2.10.4 Convert ASCII numeric character string to 8 bit integer

Entry: RDBYTE
Registers as for RDINT 2.10.3

Function: Converts an ASCII numeric character string or sub-string into an 8 bit integer

Routines: RDINT section 2.10.3

Exit: C = count on entry less number of numeric characters read
A = binary integer
HL contains address following the address of last numeric character read
Carry clear if no errors
Carry set if overflow or first character read non-numeric
BDE preserved

Calls the RDINT routine having saved DE. Checks if the converted numeric string is greater than 255 and returns.

3.1 INTRODUCTION

The NewBrain processors are provided with device driver programs which make device interfaces appear identical - all devices appear to be single byte serial. To further simplify the I/O interface a common set of calling routines, IOS, is provided to enter device drivers. Using these routines all communication with device drivers is via data streams (sometimes known as channels, logical devices or logical units). Routines are provided to initialise a device driver, assign it a data stream, allocate own memory to it, communicate with it and close it down.

The existence of a device driver is made known to IOS by the presence of its entry address in the device table. The device table is pointed to by the contents of OS fixed location DEVTAB, see section 2.8. Each device driver may consist of any number of routines but typically supports five:-

- OPENIN,
- OPNOUT,
- INPUT,
- OUTPUT,
- CLOSE

The interfaces to OPENIN and OPNOUT are identical - a physical port number and logical parameter string are passed to the routine, which is expected to initialise the device. When called via IOS the open routine may call MKBUFF in order to create own memory space. Open routines are expected to return a parameter string. The interface to INPUT, OUTPUT and CLOSE is simpler - one byte and a parameter, the port number is passed to the routine and one byte is returned. The port number is a parameter which has meaning only for the driver, and can select particular driver characteristics. For all drivers currently implemented this parameter is not used or is expected to be zero.

09NOV81-VO-CRS

NEWBRAIN-TECH-3.1 02.00

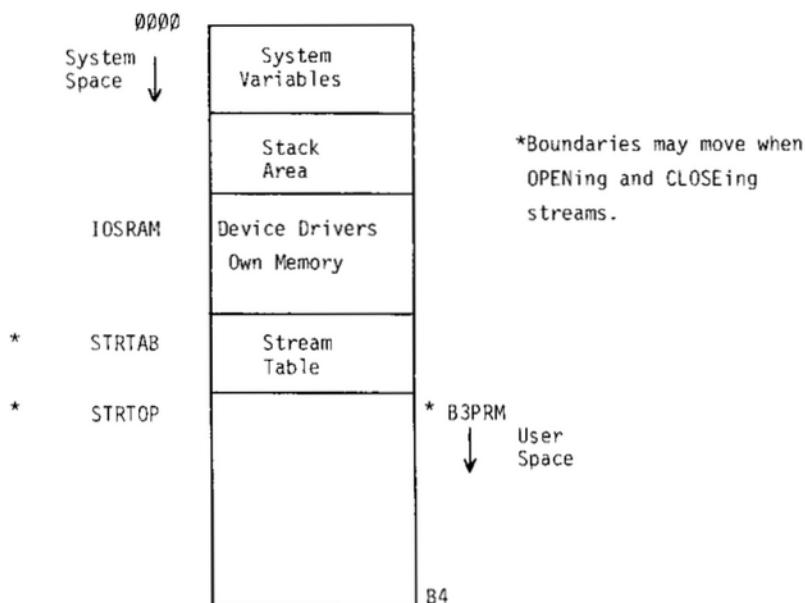
A device driver entry address is the first location of the routine table which is a list of one byte displacements to its routines. The first entry in the table defines the total number of routine entries and corresponds to a maximum request code that may be passed from IOS. A typical routine table may be defined as follows.

```
DDRIV:  DEFB 4           ; device driver entry, max request code = 4
        DEFB DOPNIN-$    ; displacement to OPENIN, request code 0
        DEFB DOPNOU-$    ; displacement to OPNOUT, request code 1
        DEFB DINP-$      ; displacement to INPUT, request code 2
        DEFB DOUT-$      ; displacement to OUTPUT, request code 3
        DEFB DCLS-$      ; displacement to CLOSE, request code 4
```

A set of six routines is provided to handle all communication between the user program (e.g. BASIC) and Newbrain devices. These routines: OPENIN, OPNOUT, INPUT, OUTPUT, MOVE and CLOSE first identify the device driver to be used and then call the appropriate routine from the device driver. OPENIN and OPNOUT allocate a stream to the device, subsequent calls to INPUT, OUTPUT and CLOSE then reference the stream rather than the driver number and port number. As mentioned above device driver open routines may call MKBUFF to allocate own memory to a device, the address of this memory is passed to the device driver on calls to INPUT, OUTPUT and CLOSE. On return from a device driver CLOSE routine the stream and the own memory are deallocated by IOS.

A recommendation is made that own memory should not contain absolute addresses. This is because CLOSEing one device may cause the own memory for another device to be moved. Some devices e.g. TV require to make adjustments to the contents of own memory should it be moved. For these the sixth routine MOVE request code 5 is implemented.

The random access memory organisation is shown in the following diagram:-



When a stream is allocated by IOS open, the stream table entry is added at (STRTOP). When own memory is allocated by MKBUFF it is added at (STRTAB) and the stream table is shifted up. The own memory entry in a stream table entry is set to zero initially and if a driver requires no own memory it should be left at zero. The own memory address may be set by the device driver to point outside the IOS data structure - if this is done it is the user's responsibility, though generally has no ill consequences for IOS. When streams and own memory are removed by CLOSE the IOS data structure is closed up.

When any of these memory adjustments take place all the IOS pointers and the own memory addresses in the stream table entries are adjusted to remain correct.

A stream table entry has the structure shown below:-

Stream Number	0
Driver Number	1
Port Number	2
Unused	3
Own Memory LS	4
Address MS	5

09NOV81-V0-CRS

NEWBRAIN-TECH-3.2 01.00

3.2 OPEN

Entry: OPENIN

OPNOUT

A = driver number

D = port number

E = stream number

BC = parameter stream length

HL points to start of parameter string

Function: Validates that no Stream Table entry is defined for input parameters, creates an entry and calls the driver OPEN routine.

Routines: FDEV section 3.8

FSTRM section 3.7

SPACE section 2.2

FDE section 3.6

Exit: Carry clear

BC, HL = parameter string returned from device driver

Carry set

A = error code from OPEN or device driver

OPEN error codes are 107, Device/Port Pair Already Open

108, Stream Already Open

100, No Space For Stream Table Entry

DE preserved

Params: PLEN *

PHPOS *

PZLEN *

STRTOP *

SAVE2 *

SAVE3 *

09NOV81-VO-CRS

NEWBRAIN-TECH-3.2 02.00

The open code executed, via the two entry points OPENIN and OPNOUT, differs only in that OPNOUT sets the carry flag on entry and OPENIN clears the carry flag. After the state of carry is fixed, the registers are pushed onto the stack and the index registers saved in SAVE2 and SAVE3 of page zero by calling the OS routine SAVIRS. If the stream number is zero indicating the console device is being opened the BASIC print formatting locations, PLEN, PHPOS, PZLEN are zeroised.

FDEV, section 3.8, is called which checks that no entry for the specified driver and port exists in the Stream Table. If it does then A is set to error code 107, Device/Port Already Open, and with carry set OPEN returns. If no entry exists, the FSTRM is called which checks that no entry for the specified stream exists in the Stream Table. If it does then it is set to error code 108, Stream Already Open, and with carry set returns to caller.

Once OPEN has determined that no Stream Table entry exists, BC is set to 6 and the routine SPACE, section 2.3, is called which obtains 6 bytes from the user program space. If insufficient memory exists then SPACE returns error code 100 with carry set and this is passed on to user by returning immediately.

STRTOP now points to offset 0 of the new Stream Table entry. The stream number, driver number and part number are written to the table entry, offsets 3,4,5 are zeroised. The new value of STRTOP, old value + 6, is set in page zero.

The state of the carry flag as set on entry through OPENIN or OPNOUT is used to determine the device driver request code. For IN this is 0, for OUT, 1. FDE, section 3.6, is called which loads HL with the OPENIN or OPNOUT routine

09NOV81-V0-CRS

NEWBRAIN-TECH-3.2 03.00

start address for the specified driver. This is swapped with the parameter string start address so that on entry to the driver OPEN routine:-

D = port number
E = stream number
BC = parameter string length
HL = parameter string start address

If an error was detected by FDE then the IOS OPEN routine returns with A containing the FDE error code without calling the driver OPEN.

On return from the driver, the index registers are restored by calling RESIRS which extract IX, IY from the temporaries SAVE2, SAVE3. If the driver OPEN routine detects an error, it will set carry and return an error code in the Accumulator. REMOVE, section 3.4, is called which in this case removes the Stream Table entry and any own memory defined by this call to IOS OPEN.

09NOV81-V0-CRSNEWBRAIN-TECH-3.3 01.00

3.3 INPUT/OUTPUT/MOVE

Entry: INPUT
OUTPUT
MOVE
A = data byte if OUTPUT
E = stream number

Function: Sets the request code in the C register and calls IOC:-

INPUT , C = 2
OUTPUT, C = 3
MOVE , C = 5

Routines: IOC section 3.5

Exit: Carry clear
A = data byte if INPUT
Carry set
A = driver error code
BCDEHL preserved

09NOV81-V0-CRSNEWBRAIN-TECH-3.4 01.00

3.4 CLOSE

Entry: CLOSE

E = stream number

Function: Sets the close request, 4, in the C register and calls the device close routine. On return removes the Stream Table entry and any own memory.

Routines: IOC section 3.5

FSTRM section 3.7

GSPCBK section 2.2

Exit: Carry may be set by device driver CLOSE routine and if this is error

A = 105, carry is cleared

BCDEHL preserved

Params: SAVE2 *

SAVE3 *

BUFLG

STRTAB*

STRTOP*

IOSRAM

TVCUR *

TVRAM *

OTHER1*

OTHER2*

DEVTAB

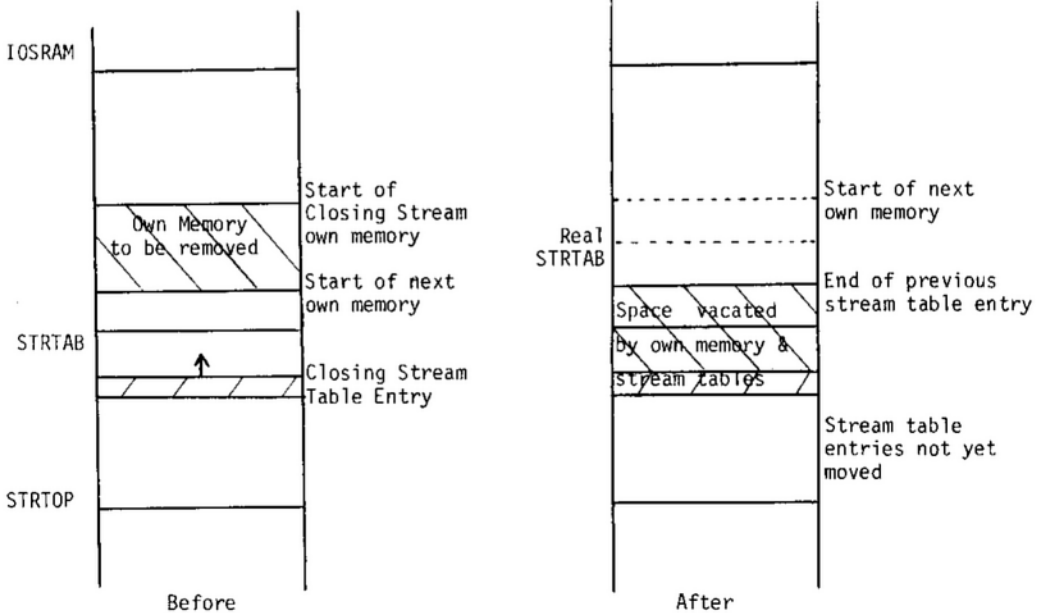
The index registers are saved in page zero via a call to SAVIRS. With the request code in C set to 4, IOC section 3.5, is called. If the driver returns with error code 105, No Stream Table Entry, the CLOSE routine returns immediately but with carry not set - error 105 is not an error for CLOSE.

For normal or any other abnormal return from IOC, the routine REMOVE is executed. This removes the Stream Table entry of the closed stream and, should there be any, the own memory for the stream. The registers AFBCDEHL are all preserved on the stack and bit 0 of BUFLG is set indicating that own memory buffers and Stream Table entries are in motion (or about to be). FSTRM is called which returns the entry within the Stream Table of the stream to be closed. CHKPBD checks that the own memory address in the entry is greater than or equal to IOSRAM and less than STRTOP. If not then IOS assumes no own memory and bypasses the own memory removal and pointer update code.

09NOV81-VO-CRS

NEWBRAIN-TECH-3.4 03.00

For an entry with own memory, the Stream Table entries higher in memory are accessed until one is found with an own memory address. Since the Stream Table entries and own memory are added in the same sequence, the next valid own memory address found in the Stream Table defines the upper bound of closing stream own memory. The difference between the start address of the closing stream own memory and the next own memory is the number of bytes to block move the remaining device drivers own memory and Stream Table entries below the closing entry.



09NOV81-VO-CRSNEWBRAIN-TECH-3.4 04.00

REMOVE then updates the following pointers by subtracting the number of own memory bytes removed:-

- STRTAB
- TVCUR
- TVRAM
- OTHER1
- OTHER2

(N.B. DEVTAB is inspected but only updated if it lies between IOSRAM and STRTOP. Should additional device drivers be added to the device table, currently in ROM, it could be moved to the IOS area between own memory and stream table entries).

The closing Stream Table entry is then removed by shifting down the higher address entries. STRTOP is adjusted by subtracting the bytes removed from device driver own memory and the Stream Table entry length, 6. This total bytes removed is given back to the user program via a call to GSPCBK. Starting with the first Stream Table entry, and accessing each entry in turn, MOVE, section 3.3, is called with HL pointing at entry start address. Returning error conditions are ignored. Once all have been processed BUFLG is cleared, the registers are popped off the stack, IX and IY are restored from SAVE2 and SAVE3, and the routine exits returning to IOS caller.

09NOV81-VO-CRS

NEWBRAIN-TECH-3.5 01.00

3.5 COMMON ROUTINE FOR INPUT, OUTPUT and CLOSE

Entry: IOC

A = data byte if called by OUTPUT

C = request code

E = stream number

Function: Recovers parameters from Stream Table entry and calls requested driver routine passing Stream Table parameters in registers.

Routines: FSTRM section 3.7

FDE section 3.6

Exit: Carry clear

A = data byte if called by INPUT

Carry set

A = error code set either by IOC or by called drive routine

IOC codes are 105, Stream (specified in E) Not Open.

DEHL preserved

The registers DE and HL are preserved on the stack. FSTRM, section 3.7, is called which finds the stream table entry matching E. If this returns with carry set then the accumulator is loaded with the error code 105, Stream Not Open, and returns to caller. On normal return from FSTRM, HL points to offset zero of the required Stream Table entry. This is incremented to retrieve the driver number into B and again to retrieve the port number into D. Offsets 4 and 5 are the own memory address for the stream and this is set in HL and pushed onto the stack. FDE, section 3.6, is called which returns with HL

09NOV81-VO-CRSNEWBRAIN-TECH-3.5 02.00

pointing at the required routine entry address. This is swapped with the own memory address at TOS so that on entry to the driver routine:-

A = data byte, if called by OUTPUT
D = port number
E = stream number
HL points to start of own memory

If an error was detected by FDE then routine returns with A containing the FDE error code and without calling the driver routine.

09NOV81-V0-CRS

NEWBRAIN-TECH-3.6 01.00

3.6 FIND DRIVER ENTRY ADDRESS

Entry: FDE

B = driver number

C = request code

Function: Computes the start address of the routine for request code
C of driver number B.

Exit: Carry clear,

HL = entry address,

Carry set,

A = 106, Invalid Device or 109, Illegal Request Code.

DE preserved.

Params: DEVTAB

HL is set with the contents of page zero location DEVTAB which is the address of the Device Table. Offset zero of the Device Table is the number of devices in the table and this is compared with the driver number in B. If B is greater than the number of devices in the table the routine returns with error code 106, Invalid Device, in the Accumulator.

Since the entries in the Device Table are two bytes long the driver entry address is defined by:-

$HL + 1 + 2*B$ since on entry HL is at offset 0 of the Device
Table and the first device address starts at
offset 1.

The two bytes starting at this computed address are set in HL which now points to the device driver entry address which is the routine table.

Offset 0 of the routine table is a number which is the maximum request code for the driver. This is compared with the request code in C and if C is greater then A is loaded with error code 109, Illegal Request Code, and routine returns to caller. Otherwise the request code is added into HL + 1 to position HL at the requested routine table entry. The value at this entry, which is the offset to the routine start location, is added into HL. If this value is zero, then the driver does not support this request code and routine returns with carry set and error 109. Otherwise the computed driver routine start address is returned in HL with carry clear.

09NOV81-V0-CRS

NEWBRAIN-TECH-3.7 01.00

3.7 FIND A MATCHING STREAM NUMBER

Entry: FSTRM

E = stream number

Function: Attempts to find an entry in the stream table to match the stream number in E.

Exit: Carry is clear, indicating a matching stream table entry is found, and HL points to offset 0 of a stream table entry.

Carry is set, indicating no stream table entry found, and HL points one past the last stream table entry (STRTOP).
ABCDE preserved

Params: STRTAB

STRTOP

The routine initialises the registers:-

- BC, AF, DE are pushed onto the stack
- the stream number is set in the accumulator
- HL is loaded with the contents of STRTAB which points to the first entry in the stream table
- DE is loaded with the contents of STRTOP which points to the location one past the last entry in the stream table.
- BC is loaded with 6, the stream table entry length.

FSTRM then enters a loop which accesses each stream table entry and compares

09NOV81-VO-CRSNEWBRAIN-TECH-3.7 02.00

the value in offset 0 of the entry with the stream number in the accumulator. For each iteration of the loop HL is incremented by BC until HL equals DE indicating the end of the stream table. If a match is found the carry flag is reset and HL is left pointing at the offset 0 of the matching stream table entry. If no match is found HL will be equal to DE itself equal to STRTOP. The registers are restored and routine returns.

09NOV81-VO-CRSNEWBRAIN-TECH-3.8 01.00

3.8 FIND MATCHING DRIVER/PORT PAIR

Entry: FDEV

A = driver number

D = port number

Function: Attempts to find a stream table entry which has a driver number and port number which matches A and D.

Exit: As for FSTRM, section 3.6

Params: STRTAB

STRTOP

The routine initialises the registers:-

- BC, AF, DE are pushed onto the stack
- HL is loaded with the contents of STRTAB which points to the first entry in the stream table
- DE is loaded with the contents of STRTOP which points to the location one past the last entry in the stream table
- BC is loaded with 6, the stream table entry length.

FDEV enters a loop which accesses each stream table entry and compares the driver number offset 1 of the table entry with the value in the Accumulator. If unequal, HL is incremented by BC to get the next entry address and the result compared with DE to determine whether all entries have been accessed. If a match is found, HL is incremented to point to the port number, offset 2 in the stream table entry. This value is compared with the required port number in D and if these agree the routine exits with carry clear and HL pointing at offset 0 of the matching stream table entry. If HL becomes equal to DE, the routine exits with carry set indicating no match found.

09NOV81-V0-CRS

NEWBRAIN-TECH-3.9 01.00

3.9 MKBUFF

Entry: MKBUFF

BC = number of bytes required

E = stream number

Function: Creates an own memory area of BC bytes.

Routines: FSTRM section 3.7

SPACE section 2.2

Exit: Carry clear,
HL points to start of own memory.
Carry set if insufficient memory,
A = 100 returned by SPACE,
BCDE preserved.

Params: STRTAB
STRTOP

SPACE is called which requests BC bytes from the user program. If there is insufficient memory, SPACE returns with carry set and A = 100. MKBUFF immediately returns to caller. If sufficient space, FSTRM is called to set HL pointing at the Stream Table entry specified by the contents of E. The offsets 4 and 5 are set to the contents of STRTAB which is the own memory address. STRTAB and STRTOP are adjusted by adding in the own memory size and the complete Stream Table is block moved BC bytes into higher memory.

02NOV81-V0-CRS

NEWBRAIN-TECH-4.1

02.00

The use of the working storage variables is as follows:-

Variable	Value
FLAGS	Bit 0 is SUPPRESS display of filenames on console during OPENIN routine. Bit 7 is the IN/OUT Flag set to 1 by OPENOUT set to 0 by OPENIN routines.
BST0	Parameter String buffer length or default 4.
BTEMP	Store for current value of B.
CTEMP	Store for current value of C. BC is the count of bytes to process and gives the current position in the data buffer.
TYPE	Really a sequence number, on OUTPUT is used to write a correct sequence number to the tape block on INPUT used to verify block sequence.
MOTOR	Contains the COP Control motor number of one of the two cassettes.

02NOV81-V0-CRS

NEWBRAIN-TECH-4.1

03.00

Each block written to tape contains the following fields:-

- block length, of two bytes is the number of data bytes in the block.
- data, of $\langle \text{block length} \rangle$ bytes
- type, one byte normally a sequence number but first block of a tape file has bit 7 set and last block has bit 6 set.
- checksum, two bytes, is the sum of all block length, data and type bytes plus an offset value 3BH.
- zeroes, nine bytes of trailer to accommodate cassette mechanism inaccuracies on stop/start.

A file is written to or read from tape by three or more calls to the tape driver. An open routine is performed - for a write a block is written to tape the filename of which is defined in the parameter string; for a read the tape is scanned until a filename block is found with filename matching the parameter string filename (if no filename is specified then the first filename block encountered is retrieved). The corresponding input or output operation is then performed by calling the relevant driver routines and transferring one character per call. On input the characters are transferred out of the buffer until the buffer empties when it is replenished from tape. On output characters are added into the buffer until full when the buffer is written to tape. The tape operation is completed by issuing a close which for input does nothing, for output writes the current contents of the buffer to tape in a short Close block.

Any attempt to deviate from this sequence will result in error e.g. attempting an input of a filename block, filename blocks can only be read by an open request.

02NOV81-V0-CRS

NEWBRAIN-TECH-4.2

01.00

4.2 OPENOUT - TP10P0, TP20P0

The value of motor number used in the cassette commands to the COP is determined by the entry point into the OPENOUT routine. The input parameter string pointed to by HL of length BC, is of the following format:

[*buffer length:][] [filename]

where "buffer length" is a string of numeric ASCII characters.

"filename" is a string of ASCII characters not starting with the ")" or "*" character.

[] indicates an optional field.

The *buffer length: field is analysed by the routine TPSYN. The existence of this optional field is determined by the presence of the * character. If the field is present the following characters up to the : character are expected to be numeric and equate to a non-zero value. TPSYN and its routines make a very thorough check of this part of the parameter string and detect the following error conditions.

- the parameter string length in BC greater than 255
- non-numeric characters between the * and :
- all zero characters between the * and :
- the character following the numerics is not a : (if no more parameter string characters follow the buffer length field then the : is not mandatory).

If none of these conditions apply the numeric value of the buffer length is multiplied by 256 to give the actual buffer length in bytes. If the *buffer length: field is absent from the parameter string then the default buffer length is 4 which when multiplied by 256 gives an actual buffer size of 1024 bytes.

The working storage requirements for the tape drives is 6 bytes which when added to the buffer size is set in BC and used as input to MKBUFF, section 3. Any errors detected either during parameter string analysis or insufficient room error from MKBUFF results in a direct return to IOS, with Carry set and the accumulator containing 136, Syntax Error in Parameter String.

On return from TPSYN, HL points to the start of own memory. At offset 0 the FLAGS variable is set with bit 7, the OUT flag as a one and if the optional parameter string field) is present bit 0, the SUPPRESS flag is set. The remaining working storage variables are set as follows:

Variable	Value
BST0	Buffer length (i.e. buffer size \div 256)
CTEMP	0
BTEMP	Buffer length
TYPE	0
MOTOR	Either 8 if entered through TP10P0 or 2 if entered through TP20P0

02NOV81-V0-CRS

NEWBRAIN-TECH-4.2

03,00

With E set to 81H indicating a Name Type data block, BC set to number of bytes remaining in the parameter string and HL pointing to the current position in the parameter string (start of filename if present), the routine WRBLCK, section 4.7, is jumped into. This writes the Name type block to tape, the data of which is the parameter string filename if present. Since WPBLCK has a RET statement at both normal and error endings control passes back to IOS after the execution of WRBLCK.

02NOV81-V0-CRS

NEWBRAIN-TECH-4.3

01.00

4.3 OUTPUT - TPOUT

On entry the Out Flag Bit 7 of Flags offset zero in own memory is tested and if not set the routine immediately returns with Carry set and the error code 135 in the accumulator - Attempting to Write to Tape File Opened for Input. The Out Flag is set by the OPENOUT routine and reset by the OPENIN routine.

The entry conditions into an output routine from IOS is that the accumulator contains the character for output and HL points to start of own memory. TPOUT does not write a data byte each time it is called but accumulates data in its own memory buffer. The pointers into the buffer are held in the working storage variables BTEMP and CTEMP. LDBUFF is called which sets up BC, D and E from the working storage variables BTEMP, CTEMP, MOTOR and TYPE respectively. The HL contents are copied into IX and HL itself is set to point to start of the own memory buffer area.

The last written position in the tape buffer is computed by adding BC to HL and the result decremented. The character in A is written to this computed location, BC is decremented and BC, D and E are saved in their respective working storage variables. This procedure has the effect of writing the first received data byte at the high address in the tape buffer and subsequent bytes in successively lower addresses, i.e. the data sequence in a tape block is in reverse order.

After saving the registers, BC is tested and if non-zero, indicating **buffer** not full, the routine returns to caller, IOS. If BC is zero then the buffer is full. BC is reset to the buffer size, E is incremented to give the data block sequence number and bit 7 of E is reset to clear the Name type indicator. The new register contents are saved in working storage and WRBLCK, section 4.7, is jumped into. This writes a data block to tape, the data of which is an entire tape buffer contents. WRBLCK returns directly to IOS.

.

02NOV81-V0-CRS

NEWBRAIN-TEXT-4.4

01.00

4.4 OPENIN - TP10PI, TP20PI

The operation of the OPENIN routines initially proceeds as for the OPENOUT routines, section 4.2. The motor number is determined by the entry point and TPSYN is called which analyses the parameter string up to the filename field and creates own memory via MKBUFF. The working storage variables are set up as follows:-

Variables	Value
FLAGS	All zero unless SUPPRESS specified in parameter string which would set Bit 0
BSTO	Buffer length defaults to 4 if the buffer length field is absent from the parameter string
CTEMP	Undefined
BTEMP	0
TYPE	Undefined
MOTOR	Either 8 if entered through TP10PI or 2 if entered through TP20PI

Having performed the initial analysis and set-up the registers are saved and with BC containing the buffer size and D set to the motor number the routine RDBLCK, section 4.8, is called. This reads in a block from tape via the COP. BC is returned as tape data block length and E as the Data Type or sequence number. If an error was detected by RDBLCK then the OPENIN routine returns to IOS with BC set to buffer size, DE set to current position in parameter string and HL pointing to the current position in the tape buffer.

02NOV81-V0-CRS

NEWBRAIN-TECH-4.4

02.00

If OK the Data Type returned in E from RDBLCK is checked to be a Name Type. If not then the routine loops back and another data block is read. If the input parameter string did not contain the filename field then any Name Type block is accepted by the routine. Otherwise the retrieved filename from the tape is compared with the parameter string filename. In all cases provided the SUPPRESS flag is not set, the retrieved filename is displayed on the console device (no error results if there isn't a stream 0) but if the input and output filenames do not match the routine loops and looks for the next Name Type block. The working storage variables TYPE and CTEMP are cleared and the routine returns to IOS with HL pointing to start of the filename and BC containing number of bytes in the filename.

02NOV81-V0-CRS

NEWBRAIN-TEXT-4.4

01.00

4.4 OPENIN - TP10PI, TP20PI

The operation of the OPENIN routines initially proceeds as for the OPENOUT routines, section 4.2. The motor number is determined by the entry point and TPSYN is called which analyses the parameter string up to the filename field and creates own memory via MKBUFF. The working storage variables are set up as follows:-

Variables	Value
FLAGS	All zero unless SUPPRESS specified in parameter string which would set Bit 0
BST0	Buffer length defaults to 4 if the buffer length field is absent from the parameter string
CTEMP	Undefined
BTEMP	0
TYPE	Undefined
MOTOR	Either 8 if entered through TP10PI or 2 if entered through TP20PI

Having performed the initial analysis and set-up the registers are saved and with BC containing the buffer size and D set to the motor number the routine RDBLCK, section 4.8, is called. This reads in a block from tape via the COP. BC is returned as tape data block length and E as the Data Type or sequence number. If an error was detected by RDBLCK then the OPENIN routine returns to IOS with BC set to buffer size, DE set to current position in parameter string and HL pointing to the current position in the tape buffer.

02NOV81-V0-CRS

NEWBRAIN-TECH-4.4

02.00

If OK the Data Type returned in E from RDBLCK is checked to be a Name Type. If not then the routine loops back and another data block is read. If the input parameter string did not contain the filename field then any Name Type block is accepted by the routine. Otherwise the retrieved filename from the tape is compared with the parameter string filename. In all cases provided the SUPPRESS flag is not set, the retrieved filename is displayed on the console device (no error results if there isn't a stream 0) but if the input and output filenames do not match the routine loops and looks for the next Name Type block. The working storage variables TYPE and CTEMP are cleared and the routine returns to IOS with HL pointing to start of the filename and BC containing number of bytes in the filename.

02NOV81-V0-CRS

NEWBRAIN-TECH-4.5

01.00

4.5 INPUT - TPINP

On entry the Out Flag, Bit 7 of FLAGS, pointed at by HL on entry from IOS, is checked, and if set the routine immediately returns with Carry set and the error code 135 in the accumulator - Attempting to Read from Tape File Open for Output. LDBUFF is called which sets up BC, D and E from the working storage variables BTEMP, CTEMP, MOTOR and TYPE respectively. IX is set to point at start of own memory and HL points at start of buffer.

If the retrieved BC is zero indicating that the buffer is empty, then a data block must be read from tape. Prior to a call to RDBLCK the type of the previous block read in is checked not to have been a Close type by testing bit 6 of the TYPE working variable. If set then the INPUT routine should not now be called but rather an OPENIN routine. Carry is set and the routine returns with the accumulator containing the error code 133 - End of Data Error.

RDBLCK, section 4.8., is called. Any errors detected causes an immediate return to IOS with Carry set. The TYPE is again retrieved and incremented. This new value is compared with the value from RDBLCK returned to E. If they agree (and the check takes into account that the Close Bit 6 may be set) then the block just retrieved from tape has a sequence number one greater than the previously read block. The data block length returned by RDBLCK if zero, i.e. a null block, causes the next block to be read.

When the buffer has some data in it, then a call to TPINP results in a character being moved from the buffer to the accumulator. Pointers into the buffer are maintained in working storage. The bytes are transferred first from a high address in the buffer pointed to by adding HL (start of buffer) and BC (characters yet to be transferred). Each time TPINP is called BC is decremented until when zero the next tape block is read.

4.6 CLOSE - TPCLS

On entry from IOS, HL points to start of own memory, in this case to the FLAGS variables. The Out Flag Bit 7 is tested and if a zero TPCLS returns immediately since on input no special action is required of the tape driver - at the end of each WRBLCK the COP is instructed with a NULLCOM command and is therefore not necessarily expecting any more cassette operations.

If open for output the close routine must write the remaining contents of the buffer to tape. LDBUFF is called which sets up the current value of BC, D and E from BTEMP, CTEMP, MOTOR and TYPE respectively. The number of bytes to be written from the buffer is computed from the formula:-

Buffer Size - BC where Buffer Size is held in BST0
BC is the current offset into the buffer.

The resultant value is set in BC as the number of bytes to write; HL is set pointing at the first valid data byte and E is incremented to give the sequence number with bit 6 of E set to indicate a close block. WRBLCK, section 4.7, is jumped into and writes a close block the data of which is the remaining data bytes in the buffer when TPCLS was called. WRBLCK returns to IOS.

02NOV81-V0-CRS

NEWBRAIN-TECH-4.7

01.00

4.7 WRITE A TAPE BLOCK

Entry: WRBLCK

BC= number of bytes to write

E = block type

HL points to start of data

Function: Write a block of data bytes to tape via the COP.

Routines: TCHR
CASSON

Exit: No special conditions, Carry set if error detected.

Params: COPCTL *
COPST *
COPBUFF *
CHKSUM *
RST8 *

02NOV81-V0-CRS

NEWBRAIN-TECH-4.7

02.00

The cassette command 80H is summed with the control bits RECORD 00H and the selected MOTOR 08H or 02H (for MOTOR1 AND MOTOR2 respectively) to give the COP command. The routine enters a loop which waits for the READY Bit 5 of COPST to go high. At this time COPST is reset to zeroes and the computed COP command set in COPCTL. With interrupts enabled the CHKSUM variable is seeded with the value of HL which happens to be the address of COPCTL i.e. 3BH. This whole operation is repeated.

The second time that the COP interrupts and the handler sets the READY, will indicate that the contents of COPBUFF (which is random) will have been written to the tape.

Data is written to tape via the COP using the routine TCHR which parenthetically is defined as RST8. The number of bytes in the data block, BC, is written to tape, using TCHR register C first. WRBLCK then enters a loop which writes the content of the address pointed to by HL to tape using TCHR, decrements BC and increments HL. This loop is exited when BC reaches zero. Following the data the contents of register E, still as on entry to the routine, is written to tape. This may contain the value 81H indicating the block is a Name type; the data block sequence number; or if bit 6 is set a Close type block.

As TCHR writes each byte to COPBUFF and waits for a COP interrupt, it additionally sums the data byte into the double precision CHKSUM and redefines itself as RST8. The checksum is written low byte first after the E register and is the sum of all written bytes starting with register C and ending with register E plus an offset equal to 3BH. As a trailer nine zero bytes are written, the COP is commanded with a null command D0H and the routine returns.

If TCHR detects an error from the COP, CERR Bit 4 of COPST set, then the error return is to WRBLCK's caller.

02NOV81-V0-CRS

NEWBRAIN-TECH-4.8

01.00

4.8 READ A TAPE BLOCK

Entry: RDBLCK

BC = Buffer Size

HL points to start of buffer area

Function: Read a block of data bytes from tape via the COP

Routines: TCHR

CASSON

Exit: BC = tape block size

E = data type or sequence number

Carry set if error. If so A contains error code

131 Length Error, data block will not fit in buffer

132 Checksum Error, computed checksum as data read in from
tape differs from tape block checksum.

130 Hardware error, returned by the COP through COPBUFF

Params: COPCTL *

COPST *

COPBUFF *

CHKSUM *

RST8 *

02NOV81-VØ-CRS

NEWBRAIN-TECH-4.8

Ø2.ØØ

The cassette command 8ØH is summed with the control bits PLAYBK Ø4H and the selected MOTOR Ø8H or Ø2H (for MOTOR 1 and MOTOR 2 respectively) to give the COP command. The routine then behaves similarly to WRBKCK - waiting for the COP to become READY, setting COPCTL with the computed COP command seeding the CHKSUM with the address COPCTL, and repeating the sequence so that the undefined byte written by WRBLCK in front of each tape block is retrieved and ignored. In fact three iterations have to be performed to retrieve this leading undefined byte:-

- first READY interrupt is received and COPCTL is set with the cassette command.
- at the time of the second READY interrupt the COP interrupt handler, section 2.5.1., sends the cassette command.
- by the third interrupt the undefined data byte will have been set in COPBUFF.

Data is read from tape, via the COP, using the routine TCHR, also used to write to tape by WRBLCK. The first two bytes retrieved by TCHR contain the block size of the data block being read. This is recovered into DE and compared with BC which an entry to RDBLCK is the size of the destination buffer. If the block size is greater than the buffer size then this results in an error code 131, Length Error, set in the accumulator and a return made to caller with Carry set. Otherwise RDBLCK enters a loop which reads DE bytes into the buffer starting at the buffer start address. When all data bytes have been read into the buffer the next byte to be recovered is the Data Type or sequence number. The checksum bytes are read via TCHR and compared with the computed value maintained by TCHR up to and including the retrieval of the Data Type byte. If these differ an error code 132, Checksum Error, is set in the accumulator and Carry is set.

The COP is commanded with the null command DØH and the routine returns.

5.1 INTRODUCTION

Three device drivers are supported by XIO:-

- TVIO, driver for a TV device
- LIIO, driver for the VF display
- TLIO, driver for concurrent TV and VF displays

One set of code modules support these drivers with different entry points for each of the devices. Generally, TVIO and TLIO operate identically and, with the exception of OPEN, the same entry points. No distinction is made between OPENIN or OPENOUT for these devices and once opened the devices may be written to or read from.

The own memory is composed of a fixed portion, the working storage and a variable portion which is the display buffer. For TV devices which have page zero EXCESS value of 24 the start address of the display buffer within own memory is required to be of the form:-



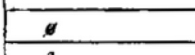
$64n + (LL/20)$ where n is an integer in the range

$1 < n < 511$

LL is the line length either 40
or 80 characters per line.

Sufficient own memory must be requested by these TV device types so that the display buffer will fit no matter what MKBUFF returns as the own memory start address. The own memory layout is as follows:-

-

Low memory address	
Offset to CSTO	0
TVMD1	1 TV Mode
-	2 Unused
-	3 Unused
DEP	4 Depth, no. of display lines held in buffer
LL	5 Line length, no. of display characters per line
EL	6 Excess + Line Length, total bytes per line
FLAGS	7 Flags
EXFLGS	8 Extra Flags
LN	9 Line Number, count of lines from top of display to current line (offset 0)
FRM	10 Frame, count of lines within buffer to first display line (offset 0)
WIN	11 Window, no. of characters in cursor line to start of 16 character VF window
INPB	12 Initial B value as start of input line
INPC	13 Initial C value as start of input line
CHAR	14 Temporary storage
DCHR	15 Temporary storage
	
For TV devices with EXCESS 24 unused locations so that TV buffer starts on $64n + (LL/20)$ boundary	
CSTO	Storage for registers C
BSTO	
ESTO	
DSTO	
FLGSTO	Flag Store
	Display buffer 1st location
	
Display buffer	
	
End bytes	
Space	
Space	
High memory address	

06NOV81-VO-CRS

NEWBRAIN-TECH-5.1

03.00

The three flag variables contain the following:

<u>Variable</u>	<u>Flag</u>	<u>Bit</u>	<u>Indicates</u>
FLAGS	SFTFLG	7	1 = Attribute on
	IFLG	6	Insert key depressed, destroyed by any control key
	CNSFLG	5	This is the console device
	CSIFLG	4	Not Used
	XYFLG	3	User sending cursor xy coordinates
	YFLG	2	0=expecting x coordinates 1=expecting y coordinate
	ESCFLG	1	ESC key has been depressed
	TVMFLG	0	Control W, the TV Mode Control has been depressed
EXFLGS	SNDCCLG	7	User has requested the character at cursor position
	SNDXYFLG	6	User has requested xy coordinates of cursor
	SNDYFLG	5	0= send x coordinate 1= send y coordinate
	CURSHOFLG	4	Show cursor
		3-0	Unused
FLGSTO	TVFLG	7	Driver supports video display
	CHRFLG	6	Characters for sending in input mode
	LIFLG	5	Driver supports VF display
	INPFLG	4	Editor in input mode
	OLFLG	3	One line display set for VF and if TV depth = 1
	PGEFLG	2	Page return mode
	HLDFLG	1	Hold mode
	CRFLG	0	Carriage Return depressed

06NOV81-V0-CRS

NEWBRAIN-TECH-5.1

05.00

All cursor and memory positional information is held as offset zero. So if the cursor is positioned at the extreme left of the first line of the current line B and C contain zeroes. GETCUR a frequently executed routine calculates the memory address of the cursor position from the expression:

$$HL + LN * EL + B * EL + C + 1$$

where HL contains the address of display buffer -1
i.e. FLGSTC address

LN is the line number of the current line in the
display buffer (offset 0)

EL is the total bytes per line, characters + excess.

So if B, C and LN are zero the result is HL + 1, the first buffer location.

06NOV81-VO-CRS

NEWBRAIN-TECH-5.2 01.00

5.2 OPEN - TVOPIN, LIOPIN, TLOPIN

The OPEN routines perform the following operations:-

- syntax checking
- own memory sizing and a call to MKBUFF
- own memory initialisation
- blanks the display

5.2.1 TVOPIN

5.2.1.1 Syntax

The routine TVSYN is called. This makes an immediate check that the Mains Present Bit 2 of the Status Register is set and if not returns to TVOPIN with error code 111, No Mains Present. The default values of DEPTH, 24, and TVLEN, 40, are defined and if the parameter string length is zero then these values are retained.

The format of the parameter string is:-

[S/L] [nnn] where S/L is the optional short/long field corresponding to 40/80 characters per line by default S.

nnn is an optional depth field which is the number of screen lines held in memory, by default 24.

06NOV81-V0-CRS

NEWBRAIN-TECH-5.2 02.00

The routine will return with a Syntax Error, code 110, if any of the following conditions are detected:-

- BC, the parameter string length greater than 255
- the first character of nnn is non numeric
- nnn evaluates to zero

The EXCESS parameter is extracted from page zero and for the 40 character per line devices is added to 40 to give the total number of bytes per line as required by the video hardware. For 80 character per line systems the EXCESS value is doubled prior to being summed with 80. The bytes per line values which may occur are 44 or 88 and 64 or 128 corresponding to EXCESS values of 4 and 24.

5.2.1.2 Sizing Own Memory

As described in the Introduction, section 5.1, the physical start addresses of TV buffer for devices with page zero EXCESS of 24 are required to be of the form $64n + 2$ for 40 characters per line and $64n + 4$ for 80 character per line systems. The actual memory requested from MKBUFF is:-

$$(90 + (\text{total bytes per line} * \text{depth})) \text{ bytes}$$

The total bytes per line * depth gives the size of the TV buffer. The 90 byte overhead is comprised of:-

- 22 bytes of working storage variables
- 64 bytes so that the TV buffer can start at a $64n + C$ address no matter where MKBUFF specifies the offset 0 byte of own memory

- 4 bytes to store the end of buffer code.

For devices with page zero EXCESS of 4, the corresponding own memory requirement is defined as:-

$$(26 + (\text{total bytes per line} * \text{depth})) \text{ bytes}$$

These devices can accept any start address for the TV buffer and consequently do not require the additional 64 byte overhead of EXCESS 24 systems.

Having computed own memory requirements, MKBUFF is called. If insufficient memory is available the OPEN routine returns to IOS immediately.

5.2.1.3 Memory Initialisation

If the device being opened is the console device, indicated by its stream number being zero, then the following page zero variables are set up:-

- PZLEN 10 for 40 character per line systems
 17 for 80 character per line systems
- PLEN set to the characters per line, 40 or 80.

The following working storage variables are then initialised.

06NOV81-V0-CRS

NEWBRAIN-TECH-5.2 04.00

Variable	Value
TVMD1	Zeroes for 40 character per line devices Bit 6 set for 80 character per line devices
DEP	Depth Value up till this time stored in D
LL	Line Length, no. of characters per line, 40 or 80 up till this time stored in B
EL	Excess + Line Length, total bytes per line, up till this time stored in C
FLAGS	CNSFLG, Bit 5 set if console device. Otherwise all zeroes.

Zeroes are written to own memory locations offset 8 to 16 that is the variables EXFLGS, LN, FRM, WIN, INPB, INPC, CHAR, DCHR. The CSTO Offset variable at offset 0 of own memory is calculated by the subroutine CALFILE. This determines the offset of the CSTO variable such that the TV buffer store, which starts +5 locations from CSTO, is set on its required memory boundary. So for 40 character per line devices with an EXCESS of 24, CALFILE calculates the next address beyond the first 16 working storage variables (the IX variables), which is of the form $64n - 3$. This value is set in own memory at offset 0. By executing GETREG, TVSYN exits having positioned on FLGSTO, offset -1 from start of TV buffer.

On return from TVSYN, IOPON, section 2.9, is called. The TVPLG Bit 7 of FLGSTO is set and if the depth value is 1, One Line Flag, OLFLG, Bit 3 of FLGSTO is set.

5.2.1.4 Display Blanking

The Clear Page control character IF is set in the accumulator and the routine OUT2 called. This is an entry point in the TVOUT routine section 5.3. This has the effect of blanking the display and positioning the cursor at the top left hand corner. On return from OUT2, HL is positioned one byte beyond the TV buffer by calling GETCR0 with the Accumulator set to the depth. 00<space><space> is written to the four locations following the TV buffer which instructs the hardware that

this is the end of TV memory. TVOPIN then returns to IOS.

5.2.2 LIOPIN

The parameter string for the VF display driver is of the form:-

nnn where nnn is a numeric ASCII character string

The string specifies the number of bytes as buffer for the VF display. The syntax checking detects the following errors which result in a Syntax Error code 110 being passed back to IOS:-

- parameter string length in BC greater than 255 bytes
- first character in string is non-numeric
- string evaluates to a number less than 16.

If no parameter string is passed to LIOPIN, BC = 0, then the default VF buffer size is 80 bytes. The registers are set up as follows:-

Register	Value
B	VF Buffer Length corresponding with line length
C	VF Buffer Length corresponding to total bytes per line
D	1, corresponding to a depth of 1
E	as on entry from IOS, the stream number

The routine LIOP is called which is an entry within TVSYN, corresponding to sections 5.2.1.2 and 5.2.1.3 which size own memory, call MKBUFF and initialise own memory. On return from TVSYN the FLGSTO variable is set with LIFLG, Bit 5 and OLFLG, Bit 3. The routine jumps to OPEN common to all drivers and described in section 5.2.1.4.

5.2.3 TLOPIN

Behaves almost identically with TVOPIN except for the flag settings in FLGSTO with the additional flag LIFLG Bit 5 set, indicating the VF display.

5.3 WRITE - TVOUT, OUT1, TVOUT

One routine performs the output for all three drivers. The entry point for TVIO and TLIO is the same and is a call to the routine TVST which enables the TV display. The next statement is the entry point for the VF driver, OUT1, which is a call to GETREG which reinstates the registers C, B, E, D from CST0, BST0, EST0 and DST0, respectively and sets HL pointing to the FLCST0 variable. The next statement is the entry point OUT2, used by the OPEN routines and is a call to OUT. On return the common exit code INEND is entered which performs the following:-

- if the device is the console, CNSFLG of FLAGS set, the page zero variable PHPOS is set to the cursor position (offset 1)
- if the CURSHOFLG of EXFLGS is set the SHOCUR routine is called which for TV devices manipulates TV1, TV2, TV3 and TVCUR of page zero so that a cursor is displayed, see section 5.4
- stores D, E, B, C in their corresponding working storage variables
- returns to IOS.

5.3.1 TVST

Calls REMCR0 which with interrupts disabled sets bit 7 of TV2, the display bit, and resets all other bits. If bit 6 of TV2 was set on entry indicating that the cursor is currently being displayed then the character at cursor position held in TV1 is set at the cursor position within the TV buffer the address of which is in TVCUR. With bit 7 of TV2 set the video is enabled at the next Frame Interrupt.

On return the page zero location TVRAM is set with the address of start of own memory.

5.3.2 GETREG

On entry expects HL to be pointing at start of own memory. At offset 0 of own memory is contained the offset to the location of the variable CST0. The values in CST0, BST0, EST0 and DST0 are set into their respective registers leaving HL pointing at FLGST0 i.e. offset -1 relative to the start of TV buffer. RST8 is seeded with the address of the GETCUR routine.

5.3.3 OUT

The flags indicating that the driver is currently sending characters are reset. These are:-

- PGEFLAG, of FLGST0, driver sending a page
- CHRFLG, of FLGST0, driver sending characters
- SNDCCFLG, of EXFLGS, driver sending character at cursor
- SNDXYFLAG } of EXFLGS, driver sending cursor position
- SNDYFLG }

If any of the receive flags are set, TVMFLG, ESCFLG, YFLG, XYFLG, indicating that the driver is expecting to receive special control characters, then the EDITA is called immediately, section 6.

Otherwise the character in the accumulator to be output is inspected and if it is the carriage return character 0D, then the following output code is called so that on return the line-feed character 0A may be set in the accumulator and the output code executed normally. For non one-line devices this means an immediate call to EDITA.

06NOV81-VO-CRS

NEWBRAIN-TECH-5.3

03.00

For one-line devices, those with OLFLG of FLGSTO set, further analysis is required:-

- if the character for output is line-feed, this is ignored and OUT returns to caller having reset CHRFLG
- if the CRFLG of FLGSTO is set, it means that the driver is waiting for a Carriage Return character as acknowledgement of the current display. If the character is CR then OUT returns immediately, if not it enters a wait loop which rejects any key other than in line cursor positioning or Newline.
- If the CRFLG is clear then the EDITA is called immediately.

The EDITA sets the character passed in the accumulator into the TV or VF buffer, or if a control character performs the necessary manipulation. A full description of XEDIT is given in section 6.

On return SETWIN is called which tests the LIFLG of FLGSTO. If this is not set, indicating the VF display does not have to be driven, then the stack is popped and a return made to OUT's caller, TVOUT.

If LIFLG is set SETWIN checks that the position of the cursor in the current line, in C, is within 15 characters of the notional start of window stored in the working storage variable WIN : WIN is the number of characters to the start of the 16 character window in the cursor line. If C is less than or equal to WIN then the contents of C are written to the WIN variable so the cursor appears at first character in window. If C is in the range $WIN < C < WIN + 15$ then no window adjustment is needed and SETWIN returns. Otherwise the WIN value is adjusted so that the cursor appears as the last character in the window unless this happens to be one character beyond the end of line in which case the window does not include the cursor.

06NOV81-VO-CRS

NEWBRAIN-TECH-5.3 04.00

On return from SETWIN, the accumulator is cleared and WR jumped into. This initially calls GETCUR through RST8 with C set to WIN which on return ensures that HL points to the first character in the 16 character window.

The value of A on entry is used to determine the offset of the cursor in the window. With A set to zero no cursor will be displayed (the COP actually sets the cursor at any character which has bit 7 set).

With HL pointing at the window position in the VF or TV buffer and DE pointing at the end of COPBUFF, the WR routine enters a loop which copies 16 bytes from the buffer into COPBUFF by incrementing HL and decrementing DE. This has the effect of writing the data to COPBUFF in reverse order.

COPST is read with interrupts disabled and if READY is set then a DISPCOM command is written to COPCTL and the routine exits back to IOS. If READY is not zero then if the command in COPCTL is DISPCOM the program routine exits to IOS; or if the command is NULLCOM, DISPCOM is set in COPCTL and the routine exits to IOS. Any other command causes the routine to loop until READY is set or COPCTL contains DISPCOM or NULLCOM.

The transfer of the data bytes from COPBUFF to the COP is performed by the COP interrupt handler, section 2.5.1.

5.4 READ - TVINP, INP1, TVINP

The TVIO and TLIO both have the same entry point which is a call to TVST, section 5.3.1. The next statement is the entry point for LIIO and is a call to GETREG, section 5.3.2.

The EXFLGS are examined to determine if a special send character is expected by the user:-

- if SNDCCFLG, bit 7 is set, then the user is expecting to be sent the character at cursor position. This is obtained by calling GETCUR through RST8 which calculates the current position and returns the character at that address in the accumulator. The special send flags are cleared.
- if SNDXYFLG, bit 6, is set, then if SNDYFLG is not set then the accumulator is set to C + 1, the screen x coordinate (offset 1). The SNDYFLG is set prior to returning to IOS.
- if SNDYFLG, bit 5, is set, then the y screen coordinate is calculated from LN + B + 1, where LN is the line number variable. The special send flags are cleared.

If any of these flags are set TVINP makes a return to IOS with the required value set in the accumulator.

If no special send flags are set the routine enters the input mode. INPFLG, Bit 4 of FLGST0, is set and XYFLG and YFLG of FLAGS are reset in case the driver was in the process of receiving cursor xy coordinates from the user. The CHRFLG is tested and if set indicates that characters are already being transferred to the user. The routine branches to the location GOTCHS, 5.4.2.

06NOV81-V0-CRS

NEWBRAIN-TECH-5.4

02.00

5.4.1 Get Characters from Keyboard

If CHPF LG is zero, then this is the first call for a character on the current input request. INPB and INPC are set with the contents of B and C which will be zero if the cursor is positioned at the extreme left of the display on the first line of the current line. If either B or C are non-zero, indicating that a prompt has been output to the current line, then the HLD FLG, Bit 1 of FLGSTO, is set disabling cursor positioning off the current line or into the prompt. TVINP then enters a loop which recovers characters from the keyboard until a Carriage Return character is detected. The operations are:-

- CALL WRC, which sets the cursor on the VF display by setting the accumulator to be the offset of the cursor position within its 16 character window, and then executing WP, described in section 5.3.3.
- CALL SHOCUR, which sets the cursor on the TV display. This sets up the following page zero variables so that on the next Frame Interrupt a cursor will be displayed:-
 - TVCUR, set to the memory address which is the current position of the cursor. This is normally the next input position unless the next input position would be on the next line in which case the cursor position is at the last input position.
 - TV1, set to the value of the character found at cursor position.
 - TV2, sets bit 6, so that cursor is displayed and if positioned at the end of a line or CRFLG set, bit 5 is set which selects the cursor blob character rather than underline.

06NOV81-V0-CRS

NEWBRAIN-TECH-5.4

03.00

- TV0, the flash clock is set to 10H so that the cursor immediately appears and remains for approximately a third of a second.
- CALL RDKEY, gets a keyboard key and converts it to an ASCII character in the accumulator, see section 7.
- CALL REMCUR, which for TV devices executes REMCR0, as described in section 5.3.1, which disables the cursor and sets the character at cursor position. This is done since EDITA expects a "clean" display buffer.
- CALL EDITA, which sets the character received by RDKEY into the display store. If the character is Carriage Return, then EDITA sets CHRFLG.
- Test CHRFLG, if zero repeats the loop.

When CHRFLG becomes set, HOLD of XEDIT is called which simply resets B and C with their original values as stored in INPB and INPC ready for transfer of characters back to the user.

5.4.2 GOTCHS

This passes a character back to the user from the current input line. CRFLG is reset, the accumulator and INPB and INPC are cleared. D and E which define the position of the last character of the current line are compared with B and C, the current position in the line. If they differ indicating more characters are to be sent to user, the memory location corresponding to the values of B and C is determined via GETCUR, and the character at that location extracted. CURRT is called which normally just increments the register pair BC but if at the end of line, but not end of the input line, B is incremented and C is set so that the continuation line character is bypassed.

06NOV81-VO-CRS

NEWBRAIN-TECH-5.4

04.00

On return from CURRT, GOTCHS jumps into the common exit code INEND.

Eventually all characters will be transferred so that BC exceeds DE on entry to GOTCHS. The flags CRFLG and CHRFLG are set, the HLDFLG is reset, and BC are cleared. A Line-Feed is output through OUT, section 5.3.3 and if the PGEFLG is not set the accumulator is set to the Carriage Return character 0D, INEND is executed. If PGEFLG is set, then if the page is completely sent, indicated by LN being zero, INEND is executed with A set to 0D after resetting PGEFLG and CHRFLG. If the page is still being sent INEND is executed immediately.

06NOV81-VO-CRS

NEWBRAIN-TECH-5.5 01.00

5.5 CLOSE - TVCLS, TVCLS1, TVCLS

The current TV display device has its start of own memory address stored in the page zero location TVRAM. If the TV device being closed has the same start of own memory address then TV2 is cleared which switches off the video after the next Frame Interrupt. For all devices TVCL1 is executed which jumps to IOPOFF, decrementing the V3 power use count in IOPUC and switching it off if IOPUC becomes zero. The IOPOFF return passes control back to IOS.

06NOV81-VO-CRS

NEWBRAIN-TECH-5.6

01.00

5.6 MOVE - RESET, Ø. RESET

Called by IOS when own memory start address changes when a stream is removed. CALFILE is called which calculates the offset to CST0 so that the TV buffer starts on the correct memory boundary. If this is the same as previously i.e. as stored at offset Ø of own memory, the RESET returns to IOS.

If the new CALFILE value differs from the old value then the CST0, BST0, EST0, DST0, TV Buffer and 4 end bytes are block moved to the new position which is set at offset Ø in own memory.

26NOV81-VO-CRS

NEWBRAIN-TECH-6.1

01.00

6.1 INTRODUCTION

XEDIT is called by XIO through the entry location EDITA. Both modules use the display own memory and the display values of BCDEHL, described in section 5.1. The edit character is passed to EDITA in the Accumulator and may be a character for insertion into the display buffer, or a control character.

Certain characters within the display buffer are used by XEDIT for control purposes:-

- ■ , the blob of value 7FH, is used as a continuation character for multi-line displays. When a functional line exceeds a display line in length the continuation character is set at the first character position of subsequent lines.
- Ø when followed by spaces to a buffer line end causes the video hardware to cease making Bus Requests for the remainder of the line. This has the effect of making the bus more available to the Z80, when the display is only partially in full, with consequent throughput enhancement.

26NOV81-VO-CRS

NEWBRAIN-TECH-6.2

01.00

6.2 MAIN LINE

On entry the character in the Accumulator is stored in CHAR. The HL pointer is incremented to point at the first location of the buffer store. If the character at this position is the line continuation character, blob, then this is set to the space character.

The HL pointer is decremented restoring it to its normal position or FLGSTO. With the edit character both in the Accumulator and CHAR the EDIT routine is called which performs the edit operation on the buffer store and the control registers, B, C, D, E. On return, the FRM variable is adjusted if necessary and the hardware switch-off code, \emptyset <space>, is set at the end of the current line.

DOFRM is called which calculates the number of lines from top of the buffer to the cursor line equal to $LN+B$. This is compared with the value of FRM which for a TV buffer indicates the start line (offset \emptyset) of the 24 line display. The following may apply:-

- $LN+B < FRM$, i.e. cursor position is above the frame; FRM is set to the value $LN+B$ so the cursor line appears as first line of frame.
- $FRM \leq LN+B \leq FRM+23$, i.e. cursor position is within the current frame; FRM is unchanged.
- $LN+B > FRM+23$, i.e. cursor position is below the frame; FRM is set so that the cursor line is last line of the frame.

On return from DOFRM, HL is saved and DELZER is called. This calls GETCUR with B, C set to D, \emptyset so that HL points to first character of last line of current line.

26NOV81-VO-CRS

NEWBRAIN-TECH-6.2

02.00

If E is equal to the line length LL then the routine returns. Each character back from the line end to the Eth character is checked to be a zero. If no zero is found the routine returns, otherwise the zero is replaced by a space character. This is to remove any zeroes left behind by the EDIT operation e.g. character deletion.

Unless the last line of the current line is full, the greater of E or C is added to the line start address and a zero written to the resulting location. With HL pointing to FLGSTO and B, C, D, E as modified by the EDIT function the program returns to caller.

6.3 EDIT

The EDIT routine is called with the edit character in the Accumulator and CHAR. The value of flag settings and the edit character are tested to determine the edit function to be performed:-

- if XYFLG set, indicates that editor expects a cursor coordinate edit character. Section 6.3.1.
- if TVMFLG set, indicates that editor expects a TV mode value. Section 6.3.2.
- if ESCFLG set, indicates that whatever the value of the edit character this should be inserted into the buffer store. Section 6.3.3.
- if the edit character is not a control character then it is to be inserted into the buffer store. Section 6.3.3.
- if HLDFLG and OLFLG not set, indicates that all control characters are valid, so the control request in the Accumulator is actioned. Section 6.3.4.
- If HLDFLG or OLFLG set, indicating cannot move off current line or one line display respectively, then certain control codes are invalid and others are translated into one line equivalents:-

26NOV81-V0-CRS

NEWBRAIN-TECH-6.3

02.00

- IL, 1H, Insert Line
 - DL, 2H, Delete Line
 - MP, 12H, Make new Line
 - MS, 13H, Make continuation line
 - CLEAR, 1FH, Clear Page is converted to CLL, 1CH, Clear Line
 - FF, CH, Home is converted to CHL, 1CH, Cursor Home Left.
- are invalid
converted to NULL

The resultant code is processed. Section 6.3.4.

Unless specified the CRFLG is always reset after the EDIT operation.

6.3.1 Set Cursor Position, SETXY

The control code 16H sets the XYFLG, Bit 3 of FLAGS. EDIT expects the next two characters received to be the x and y coordinates of the cursor offset 1, i.e. the x, y coordinates of the screen top left hand cursor is sent as 1, 1. If YFLG is set then the x coordinate has already been received and the current character in the Accumulator is the value of the y coordinate.

If YFLG is not set, the contents of the Accumulator are saved on the stack and the routine HOME called. This sets B and C to zero. The Accumulator contents are restored. If A equals zero then the routine returns immediately with carry clear. In this case a coordinate of zero is considered as a coordinate of one. A is decremented (to offset 0), compared with the line length and if greater set to the line length. The resultant value is set in C. On exit from SETXY the YFLG is set.

On the second pass through SETXY, the YFLG will be set and the character in the Accumulator is the cursor y coordinate. The XYFLG and YFLG are both reset.

26NOV81-V0-CRS

NEWBRAIN-TECH-6.3

03.00

The contents of the Accumulator are tested and if zero the routine returns. The routine will also return immediately if the value of the y coordinate is 1 corresponding to the top line of the display buffer. Otherwise LNFD is called A-1 times which line feeds to the required y coordinate incrementing the B register each time it is called.

6.3.2 Set TV Mode, TVMSET

The control code 17H sets the TVMFLG, Bit 0 of FLAGS, for Video devices. EDIT expects that the next character received is a TV mode control byte and this is set in the own memory working storage variable TVMD1. The contents of TVMD1 are output to the TV Control Register, port 12, at Frame interrupt time via the XEDIT routine SETFRM. This routine also computes the start address within the display buffer from which the video hardware is to start accessing. This is set in BC.

6.3.3 Insert Character in Display Buffer, INSERT

The character in the Accumulator and copied in the CHAR location is to be inserted into the display buffer at the cursor position specified by B and C. On entry all the IX flags except SFTFLG in FLAGS are set to zero. If the SFTFLG is set then the bit 7 of the character passed to EDITA is complemented. If the resulting character is 00H, EDIT returns immediately. The current value of B and C are checked to be within or beyond the current line and not in the INPUT prompt area. If not carry is set and EDIT returns. The insert flag IFLG is tested which is set when the control code 11H has been passed to EDIT. If set the routine branches to INSNOW, section 6.3.3.1.

The CRFLG is tested. This is set by X10 when all characters have been passed to the User in input mode, section 5.4.2. If CRFLG is set then it indicates to the editor that the cursor is positioned at the start of a line and that that line must be deleted from the buffer to make way for the first character of a new current line.

26NOV81-V0-CRS

NEWBRAIN-TECH-6.3

04.00

DLIN is called which deletes the current line and any continuation lines that may exist. CPDB is called which returns with carry clear if BC is beyond the current line or carry set if within the current line. If carry clear then E is set equal to C marking the new end of the current line and INCDE called. This increments E the character count of the last line of the current line and conditionally increments D if E is equal to the line length.

CURRT and CURLT are called. The reason for this seemingly worthless operation is to position the cursor correctly when inserting the first real character of a continuation line. The cursor at this time is positioned on the last character of the previous line. CURRT positions it two positions beyond the blob character and CURLT moves it adjacent to the blob. The actual address of this location is computed using GETCUR and the character in CHAR inserted. CURRT is executed and EDIT returns.

6.3.3.1 Insert Mode, INSNO

IFLG is set as a result of depressing the INSERT keyboard key. In this mode characters entered within the current line cause all others to the right and below the cursor position to be shifted to the right. CPDB is called which determines whether the cursor is within or beyond the current line. If beyond then E is set equal to C marking the new end of the line. INCDE is called which increments the total character count for the line. Starting with the cursor line the characters to the right of the cursor are moved to the right. HL is set pointing to the cursor position location via GETCUR. The character at this location is retrieved into the accumulator and the input character now in C is written at HL. HL is incremented, the accumulator contents are moved to C and the process repeated until the end of line is reached.

This operation is repeated for each line beyond the original cursor line with B incremented each time and C set to 1 bypassing the continuation character. CURRT is executed and EDIT returns.

26NOV81-VO-CRS

NEWBRAIN-TECH-6.3

05.00

6.3.4 Control Functions

The character in the Accumulator is set in CHAR since the control character in A may have been modified by EDIT, see section 6.3. A table of control routines EDADD is indexed by A and the control routine executed. On completion of the control function the control routine returns to EDIT's caller.

The following subsections detail the operation of each control routine. The control functions are summarised in the Table 6.3.4, overleaf.

6.3.4.0 0H, Control @, NULL, Null

Does nothing, returns immediately.

6.3.4.1 1H, Control A, INSLIN, Insert Line

INSLIN uses the general insert line routine ILIN which makes space for a line to be added to the current line. On entry to INSLIN, ABCDE are zeroised and A is set to the value of LN, prior to the execution of ILIN. The general entry condition for ILIN is that the Accumulator contains the line number offset 0 which is to be vacated making way for the line yet to be inserted. If the line to be vacated is the last line on the display then all lines are scrolled up and the first line is lost. If not the last line of the display then the line to be vacated and all lines below it are scrolled down and the last line is lost.

26NOV81-V0-CRS

NEWBRAIN-TECH-6.3

06.00

Table 6.3.4

<u>HEX.</u>	<u>DECIMAL</u>	<u>CONTROL</u>	<u>KEY</u>	
0	0	@		Null
1	1	A	Sh/insert	Insert line
2	2	B	Sh/+	Delete Line
3	3	C	Cntl/newline	Send page
4	4	D		End of file
5	5	E		Send line
6	6	F		Show cursor
7	7	G		Cursor off
8	8	H	←	Cursor left
9	9	I	Cntl/escape	Tab 8 spaces
A	10	J	↓	Cursor down
B	11	K	↑	Cursor up
C	12	L	Home	Cursor home
D	13	M	Newline	Newline
E	14	N	Sh/+	Attribute on
F	15	O	Sh/escape	Attribute off
10	16	P		Graphics escape
11	17	Q	Insert	Enter insert mode
12	18	R	Grph/+	Make new line
13	19	S	Grph/+	Make continuation line
14	20	T	Grph/newline	Send cursor character
15	21	U	Grph/insert	Send x, y
16	22	V	Cntl/insert	Set cursor x, y
17	23	W	Grph/escape	Set TV control
18	24	X	Sh/←	Delete left
19	25	Y	Sh/→	Delete character
1A	26	Z	→	Cursor right
1B	27	[Escape	Escape next character
1C	28]	Cntl/←	Cursor home left
1D	29	\	Cntl/→	Cursor home right
1E	30	+	Cntl/home	Clear line
1F	31	-(£)	Shift/home	Clear page

26NOV81-V0-CRS

NEWBRAIN-TECH-6.3

07.00

6.3.4.2 2H, Control B, DELLIN, Delete Line

SCROLL is called D+1 times with A set to LN. SCROLL moves lines below that specified in the Accumulator up one line. The functional line in which the cursor is positioned on entry is thus removed. The line immediately below the deleted line becomes the current line, BC contains zeroes and D and E are computed for this new line by executing LNFD4, which counts the number of continuation lines and the number of characters in the last line.

6.3.4.3 3H, Control C, PGERET, Send Page

If HLDFLG is set returns immediately otherwise sets PGEFLG. LN is set to zero and the D and E values of the first line computed. Subsequent calls for input through XIO will return the whole functional screen unless cancelled by a call to output.

6.3.4.4 4H, Control D, NULL, End of File

Does nothing, returns immediately.

6.3.4.5 5H, Control E, LNERET, Send Line

Subsequent calls to input will return the current line.

6.3.4.6 6H, Control F, SHOWC, Show Cursor

Sets the CURSHOFLG and returns.

6.3.4.7 7H, Control G, HLDEC, Cursor Off

Resets the CURSHOFLG and returns.

6.3.4.8 8H, Control H, CURLT, Cursor Left

If the LIFLG is not set then B and C are compared with INPB and INPC which mark the start of input position of the current line. If B and C are equal to INPB and INPC respectively then the cursor cannot move further left so routine returns. For single line displays the cursor may move left to inspect the prompt but data may not be input.

The cursor is moved left by decrementing C, decrementing B if C becomes 1 on a continuation line. In this case C is set to LL-1.

6.3.4.9 9H, Control I, DOTAB, Tab 8 spaces

The cursor is positioned to the right such that C contains a value divisible by 8. CURRT is called sufficient times so that this occurs. e.g. if C=1, CURRT is called 7 times after which C=8.

6.3.4.10 AH, Control I, PLNFD, Cursor Down

If in input return mode, CHRFLG set, then on return from EDIT the CRFLG is not reset and the line feed operation of Cursor Down is considered as occurring from the last line of the current line. If CHRFLG flag is not set, then the same processing occurs if B equals D. Otherwise B is just incremented and if the line feed would move into a continuation character C is incremented.

When moving from the last line of a functional line into a new line, the HLDFLG is tested because if set the cursor cannot move out of the current line and routine returns. The value of D+LN+1 is computed and if equal to the depth, DEP, the routine returns, i.e. jams on the last line of the display buffer, but if CRFLG is set then whole screen scrolls. If D+LN+1 is not equal to the depth, then this computed value is set as the new value of LN.

The new current line about to be entered is sized by counting the number of continuation characters and the number of characters on the last line. This results in the setting of D and E.

6.3.4.11 BH, Control K, CURUP, Cursor Up

Behaves inversely to Cursor Down. If B is zero on entry then moving up will enter a new line. This is sized to set D and E. If B is not zero then it is just decremented. If this results in it becoming zero then C is tested and if it contains 1 it is decremented. The cursor is not allowed to move up into the prompt area and if this would result the cursor is positioned at the position immediately following the prompt.

6.3.4.12 CH, Control L, HOME, Cursor Home

BCDE are zeroised. The line that exists at the display buffer start is sized to set D and E.

6.3.4.13 DH, Control M, NLINE, Newline

Sets the end of line flags and returns such that CRFLG is not reset. CRFLG and CHRFLG are set and HLDFLG is cleared. B and C are zeroised and routine returns. If CRFLG was set on entry, as will occur if DH DH is passed to the EDITA, then the condition of INPFLG determines processing performed.

- INPFLG = 1, D and E are set into INPB and INPC
B and C are set to zero
- INPFLG = 0, the complete line following the line Newline'd
out of is deleted.

-

6.3.4.14 EH, Control N, SHFTIN, Attribute On

Sets SFTFLG which causes the bit 7 of all subsequent non-control codes to be inverted before displaying.

6.3.4.15 FH, Control O, SHFTOU, Attribute Off

Clears SFTFLG set by SHFTIN in previous section.

6.3.4.16 10H, Control P, NULL, Reserved for Graphics Escape

At present does nothing, returns immediately.

6.3.4.17 11H, Control Q, INSMOD, Enter Insert Mode

Sets IFLG and returns.

6.3.4.18 12H, Control R, MKPRM, Make New Line

Makes the current physical line the first line of a functional line. This is a null operation if the cursor line is already a first line with B equal to zero. Effected by making B equal to 0 and C equal to 1 and performing a DELLT, section 6.3.4.24, effectively deleting the continuation character.

6.3.4.19 13H, Control S, MKSEL, Make Continuation Line

Makes the current physical line a continuation line of the previous functional line. This is a null operation if the cursor line is already a continuation line i.e. if B not equal to zero.

26NOV81-VO-CRS

NEWBRAIN-TECH-6.3

11.00

6.3.4.20 14H, Control T, CCRET, Send Cursor Character

If INPFLG is set then this control request is ignored. Otherwise, the SNDCCFLG is set which is intercepted by XIO and sends the character at cursor on a subsequent call for input, section 5.4.

6.3.4.21 15H, Control U, XYRET, Send Cursor x, y

If INPFLG is set then this control request is ignored. Otherwise the SNDXYFLG is set which is interrupted by XIO and sends the x and y coordinates on subsequent calls for input, section 5.4.

6.3.4.22 16H, Control V, XYRQST, Set Cursor x, y

If the HLDFLG is set then this control request is ignored. Otherwise the XYFLG is set. The subsequent two characters passed to EDITA are expected to be cursor x, y coordinates, section 6.3.1.

6.3.4.23 17H, Control W, TVRQST, Set TV Control

For video devices the TVMFLG is set. The subsequent character passed to EDITA is set in TVMD1, section 6.3.2.

6.3.4.24 18H, Control X, DELLT, Delete Character to left of Cursor

If the cursor is within or immediately to the right of the prompt area the routine returns immediately. For one line displays if the character to be deleted cannot be viewed since C equals WIN the routine returns without deleting.

CURLT, section 6.3.4.8, is performed. The characters to the right and below the new position of the cursor are shifted one position to the left. If this results in the last line of the current line becoming empty, lines lower in the display buffer are scrolled up one line.

26NOV81-V0-CRS

NEWBRAIN-TECH-6.3

12.00

6.3.4.25 19H, Control Y, DEL, Delete Character at Cursor

An entry point within DELLT, this operates similarly but does not perform the cursor left via CURLT.

6.3.4.26 1AH, Control Z, CURRT, Cursor Right

If C is less than LL then it incremented and routine returns. If B is greater than equal to D then neither B or C are incremented and cursor remains at line end. Otherwise B is incremented and C is set beyond the continuation character of the next line.

6.3.4.27 1BH, ESCAPE, ESCAPE, Escape next Character

Sets the ESCFLG so that the next character passed to EDITA is set in the display buffer regardless of value.

6.3.4.28 1CH, Control ←, HOMEL, Cursor Home Left

B and C are set to the first input position of the current line. For one line displays B and C are always set to zero; for other displays B and C are the values contained in INPB and INPC.

6.3.4.29 1DH, Control →, HOMER, Cursor Home Right

C is set to LL and B is set to D.

6.3.4.30 1EH, Control HOME, DLINE, Delete Line

Deletes the current line excluding the prompt. B and C are loaded from INPB and INPC. If B is not equal to D then routine loops calling SCROLL with the Accumulator equal to D+LN each time deleting D. This deletes all lines below the cursor line.

26NOV81-V0-CRS

NEWBRAIN-TECH-6.3

13.00

WLIN is called which writes a zero at the cursor position and spaces to the remainder of the display line, which causes the hardware to stop making bus requests for remainder of the line.

6.3.4.31 IFH, Shift HOME, CLRSCR, Clear Page

Calls WLIN for each display buffer line with C equal to zero. This sets a 0 followed by LL-1 spaces in each line causing the hardware to stop making bus requests for remainder of the line. HOME, section 6.3.4.12, is called which zeroes B and C and sizes the first line in the buffer, which in this case is of zero length.

26NOV81-VO-CRS

NEWBRAIN-TECH-6.4

01.00

6.4 XEDIT ROUTINES

This section contains routines which are either useful or demonstrate some operations performed by XEDIT. Unless specified all routines expect BCDEHL to be in their display format and display own memory to be correctly defined.

6.4.1 ADHLA, Add A to HL

Function: $HL = HL + A$

Preserves: BCDE

6.4.2 BACAL, Calculate A times D

Function: $BC = A * D$, D must be greater than zero on entry.

Preserves: EHL

6.4.3 CALIN, Calculates display buffer offset to a location preceding a line of interest

Entry: A = a display line number offset 0

Function: $BC = EL * A$

Preserves: ADEHL

6.4.4 ADLIN, Computes address in display buffer of first location of a line

Entry: A = a display line number offset 0

Function: $HL = HL + EL * A + 1$

Preserves: ABCDE

26NOV81-V0-CRS

NEWBRAIN-TECH-6.4

02.00

6.4.5 ADLIN1, Computes address in display buffer of location preceding a line

Entry: A = a display line number offset 0

Function: HL = HL+EL*A

Preserves: ABCDE

6.4.6 GETDE, Computes the address one line on

Entry: HL points within the display buffer

Function: DE = HL on entry
HL = DE+EL

Preserves: BC

6.4.7 GETCUR, Computes the address in the display buffer of the cursor

Function: HL = HL+EL*(LN+B) + C+1
and A contains the character at this location

Preserves: BCDE

26NOV81-VO-CRS

NEWBRAIN-TECH-6.4

03.00

6.4.8 GETCRØ, Computes a cursor initial address

Entry: A = a display line number offset Ø

Function: $HL = HL + EL * A + C + 1$
and A contains the character at this location.

Preserves: BCDE

6.4.9 CPDB, Compares DE with BCFunction: Carry set if within the current line
Carry clear if beyond the current line

Preserves: BCDEHL

6.4.10 INCDE, increments DE

Not a particularly useful routine but of interest demonstrating the type of operations that are performed by XEDIT which supports the one line and multi-line display types. This routine is called in INSERT prior to inserting the character into the display buffer at the cursor position.

On entry the value of E is compared with LL. If less than LL, E is incremented and the routine returns. D is incremented and compared with the depth in DEP. If these values are the same then the current line has filled the display buffer. INPFLG is tested and if set the routine immediately returns and no more characters may be input if this full display buffer condition is reached. If INPFLG is zero, implying output mode then OLFLG is tested. For multi-line displays the output is allowed to continue though the display will scroll upwards. For one line displays the current output character is saved and a Newline character passed to the EDITA through XIO.

26NOV81-V0-CRS

NEWBRAIN-TECH-6.4

04.00

On return the current output character is restored and XIO is jumped into such that the Newline wait loop is entered, section 5.3.3, which waits for a Newline character to be entered before proceeding.

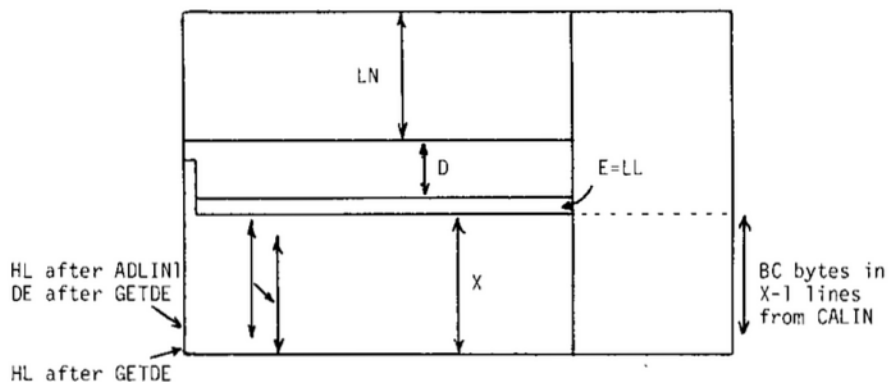
When display buffer is not full but the current input line is full the routine ILIN is called. This uses a number of the routines described in previous sub-sections. One of two procedures may occur depending on the position of the just filled line.

26NOV81-VO-CRS

NEWBRAIN-TECH-6.4

05.00

If there are lines below this line then these are moved down and the last line in the buffer is lost.



On entry to ILIN, the Accumulator contains $D+1+LN$ which is the line number, offset 0, of the line to be vacated. X is calculated to be $DEP-D+1+LN$. With $A=X-1$, CALIN computes $BC=EL*(X-1)$. With $A=DEP-1$, ADLIN1 is called which calculates the start address of the last line but one in the display buffer. GETDE puts this address in DE and the start address of the last line in HL. DE and HL are exchanged and an LDDR performed vacating the line following the current line. WLIN is called which writes a zero to the start location of the vacated line and $LL-1$ spaces following, thereby switching off the video hardware for the duration of that line.

If the just filled line is the last line, i.e. $X=0$, then a similar operation is performed which scrolls the last $DEP-1$ lines up the buffer thereby vacating the last line and losing the first. WLIN is called as before.

On return from ILIN the continuation character is set at the first location of the new line and E is set to 2.

7.1 INTRODUCTION

The keyboard driver module supports two devices:-

- KBWIO, keyboard wait for a key
- KBIIO, keyboard immediate return with key if available.

The two devices differ only on INPUT and share common OPEN, OUTPUT and CLOSE routines. No distinction is made between OPENIN and OPENOUT, either is allowed and opens the stream for input or output with an own memory allocation of one byte.

The keyboard driver recovers a keyboard matrix coordinate from one of two routines dependent on the device:-

- GETKEY, for KBWIO
- IMMKEY, for KBIIO

The returned keyboard key is converted into a value which is used to access the character ROM. The converted value may lie in the range 0H to FEH.

	0	- 1FH	Control Codes
	20	- 3FH	ASCII numerics and symbols
A	40	- 5EH	ASCII upper case letters and symbols (Group A)
a	5F	- 7FH	ASCII lower case letters and symbols
-	80	- 7EH	Graphics
α	A0	- BEH	Secondary Graphics
Δ	C0	- DEH	Tertiary Graphics
ζ	E0	- FEH	Shifted Graphics

The graphics are all derived from the Group A values as shown for the letter A.

18NOV81-VO-CRS

NEWBRAIN-TECH-7.1 02.00

The parameter KBMODE influences the returned character value. The least significant four bits of KBMODE are important:-

<u>Bits</u>	<u>Value</u>
0	Shift lock, when set converts lower case alphas to upper case.
1	Graphics shift
2	Secondary Graphics select
3	Tertiary Graphics select

The routine KLOOK, section 7.7, which used KBMODE only tests Bit 3 if Bit 2 is set, i.e. both must be set to obtain Tertiary Graphics. KBMODE is set up by RDKEY. When the control key and one of the numeric keys 0 to 9 is depressed, the corresponding bits of KBMODE are set as follows:-

<u>Mode</u>	<u>Bits 3-0 of KBMODE</u>	<u>KBMODE</u>
0	0000	90H
1	0001	91H
2	0010	92H
3	0011	93H
4	0100	94H
5	0101	95H
6	0110	96H
7	0111	97H
8	1110	8EH
9	1111	8FH

18NOV81-VO-CRSNEWBRAIN-TECH-7.2 01.00

7.2 OPEN - KBOPI

The open routine KBOPI is used by KBIIO and KBWIO. It calls MKBUFF with BC set to 1. The one byte of own memory is set to zero. BC is set to zero prior to return indicating a null length returned parameter string.

18NOV81-VO-CRS

NEWBRAIN-TECH-7.3 01.00

7.3 INPUT - KBIINP, KBWINP

The two input routines differ only in the "get-key" call. KBIINP makes a call to IMMKEY which returns immediately with a keyboard key if one is ready, null if not; KBWINP calls GETKEY which waits until a keyboard key is depressed. Both IMMKEY and GETKEY are described in section 2.4. Prior to the appropriate call both routines swap the contents of own memory with the contents of the page zero location KBMODE. For KBIINP if no keyboard key is returned then the contents of KBMODE and own memory are swapped back again and routine returns with zeroes in the accumulator.

Otherwise, and always for KBWINP, the routine KLOOK is called, section 7.7 which converts the keyboard matrix coordinate returned by the "get-key" call into an ASCII or graphics character.

On return from KLOOK the KBMODE contents and own memory are swapped back. If carry was set on return from KLOOK the the accumulator is cleared prior to return to IOS. Otherwise the ASCII or graphics character is returned.

18NOV81-VO-CRSNEWBRAIN-TECH-7.4 01.00

7.4 OUTPUT-KBOUT

The function of the keyboard driver output routine is to set up own memory with the KBMODE shift status. Provide the value in the accumulator is in the range 0-9 then the contents of the accumulator are written to own memory. On input the contents of own memory are swapped with KBMODE; KBMODE is then read to determine shift and graphic lock status.

18NOV81-V0-CRSNEWBRAIN-TECH-7.5 01.00

7.5 CLOSE-KBCLS

Merely clears carry and returns.

18NOV81-V0-CRSNEWBRAIN-TECH-7.6 01.00

7.6 RDKEY

Entry: RDKEY

Function: Waits for a valid keyboard key to be depressed and converts to an ASCII or graphics character.

Routines: GETKEY section 2.4
KLOOK Section 7.7

Exit: A = character
Carry is always clear
BCDEHL preserved

Params: KBMODE*

The registers are preserved on the stack and GETKEY unconditionally called through the intercept mechanism. On return KLOOK is similarly called. Three exit conditions from KLOOK are considered:-

- Carry clear, registers are reinstated and routine exits
- Carry set and zero flag set, in this condition KLOOK returns HL pointing at page zero location KBMODE. The contents of the accumulator returned by KLOOK are written to the KBMODE location and routine loops back to GETKEY call.
- Carry set and zero flag set, routine loops back to GETKEY call since keyboard keys depressed has requested an illegal graphics character.

18NOV81-VO-CRSNEWBRAIN-TECH-7.7 01.00

7.7 KLOOK

Entry: A = keyboard matrix coordinate

Function: Convert keyboard matrix coordinate to ASCII or
graphics character

Routines: TTCAPS section 2.10

Exit: Carry clear
A = character
Carry set and Zero Flag set = mode change request
Carry set alone = invalid keyboard key combination

Params: KBMODE

The lower six bits of the keyboard matrix coordinate determine the keyboard key depressed, the upper two bits determine whether either the Shift, Control or Graphics key was depressed simultaneously. On entry to KLOOK the lower six bits are extracted by ANDing the accumulator contents with 3FH. If any of the top two bits were set on entry then 40H is added to this result. The resultant value is used to extract a character from a table, into the accumulator. The table contents are shown overleaf.

18NOV81-VO-CRS

NEWBRAIN-TECH-7.7 02.00

Bits 3-0	Bits 6-4							
	000	001	010	011	100	101	110	111
0	ØH	ØH	ØH	ØH	ØH	ØH	ØH	ØH
1	ØH	ØH	ØH	ØH	ØH	ØH	ØH	ØH
2	BS	HT	LF	VT	ØH	1H	2H	3H
3	7	8	u	j	97H	8EH	U	J
4	6	9	i	n	96H	8FH	I	N
5	5	Ø	y	m	95H	90H	Y	M
6	4	(o	,	94H	[.	O	4H
7	3)	z	.	93H]	L	5H
8	2	*	;	b	92H	98H	6H	B
9	1	VD	h	v	91H	7H	H	V
A	t	p	g	c	T	P	G	C
B	r	=	f	x	R	@	F	X
C	e	-	d	z	E	\	D	Z
D	w	+	s	INS	W	Λ	S	8H
E	q	CR	a	k	Q	9H	A	K
F	space	ESC	/	FF	AH	BH	CH	DH

N.B. Bit 6 set if any of Graphics, Shift or Control keys depressed

First Conversion Look-Up Table

A particular group of characters, @, A to Z, [,\,], A, are used to drive the four sets of graphics characters. This group of 31 original upper case characters is referred to as Group A. All other values extracted from the table above are not Group A. The subsequent processing is determined by the condition of the two shift bits, 7 and 6, of the accumulator on entry. The following states can exist:-

<u>Bits 7&6</u>	<u>Value</u>
00	Unshifted
01	Upper case shift key
10	Control key
11	Graphics key

Note that if any of the shift bits are set then the value extracted from the table will be predominantly group A characters. The remaining values in the right hand four columns of the table are used as entries into sub-tables.

7.7.1 Shift Bits = 00, Unshifted

Bit 0 of KBMODE is examined. This is the shift lock bit. If set then a call is made to the OS routine TTCAPS, section 2.10, which will convert the value in the accumulator to its upper case equivalent provided it is one of the 26 lower case letters. Carry is cleared prior to returning.

18NOV81-VO-CRSNEWBRAIN-TECH-7.7 04.00

7.7.2 Shift Bits = 01, Shift Key

For group A characters the Graphics Shift Bit 1 of KBMODE is examined. If set then A0H is added to the ASCII value of the group A character in the accumulator to give the shifted graphics set of characters E0H to FEH. The routine returns with carry clear.

For non-group A characters, Bit 7 of the accumulator is reset and the resultant value indexes the sub-table overleaf extracting a character into the accumulator. Carry is cleared and routine returns. E.g. Shift 1 will give 91H from first table, resetting Bit 7 gives 11H which indexes "!" from Sub-Table 1.

18NOV81-VO-CRS

NEWBRAIN-TECH-7.7 05.00

<u>Value</u>	<u>Character</u>	<u>ASCII</u>	<u>Effect</u>
0	RUBOUT	18H	Delete left
1	RUBRT	19H	Delete character
2	DL	2H	Delete Line
3	SO	EH	Attribute On
4	<		
5	>		
6	:		
7	_		
8	IL	1H	Insert Line
9	CR	DH	Newline
A	Space		
B	SI	FH	Attribute Off
C	?		
D	CLEAR	1FH	Clear page
E	(
F)		
10		C0H	
11	!		
12	"		
13	#		
14	\$		
15	%		
16	&		
17	'		
18		E4H	

Sub-Table 1

7.7.3 Shift Bits = 10, Control key

For group A characters, 40H is subtracted from the ASCII value in the accumulator and routine returns with carry clear.

For non group A if bit 7 of the accumulator is set then routine returns with carry and zero flag set. This feature is specifically used by RDKEY, section 7.6, to set the KBMODE parameter in page zero. This condition only happens for the numeric keys when depressed concurrently with the control key. See the Introduction 7.1. For non group A characters which do not have bit 7 set then the value in the accumulator is used to index the table overleaf, returning with the extracted value in the accumulator and carry clear.

E.g. Group A character W = ASCII 57H; subtract 40H gives
17H which is Control W.

Control. gives 5H from the first conversion table which indexes
; from Sub-Table 2.

18NOV81-VO-CRS

NEWBRAIN-TECH-7.7

07.00

Value	Character	ASCII	Effect
0	CHL	1CH	Cursor home left
1	CHR	1DH	Cursor home right
2		0H	
3		0H	
4		7FH	
5	!		
6		C1H	
7		0H	
8	XY	16H	Set cursor x y
9	PR	3H	Send Page
A	Space		
B	TAB	9H	Tab 8 spaces
C		C2H	
D	CLL	1EH	Clear Line

Sub-Table 2

7.7.4 Shift Bits = 11, Graphics Key

For group A characters, 40H is added to the ASCII value in the accumulator selecting the ordinary graphics. The KBMODE Bit 3 is tested. If set the secondary graphics are selected by adding a further 20H to the value in the accumulator. If Bit 4 of KBMODE is also set then another 20H is added to the accumulator selecting tertiary graphics. For each of these graphics selection the routine returns with carry clear.

For non group A, if bit 7 of the accumulator is set then routine returns with carry set. If bit 7 is not set then the value in the accumulator indexes the table overleaf. The value extracted is set in the accumulator and routine returns with carry clear.

e.g. Graphics. gives 5H from the first conversion table which indexes } from Sub-Table 3.

18NOV81-V0-CRS

NEWBRAIN-TECH-7.7

09.00

Value	Character	ASCII	Effect
0		0H	
1		0H	
2	MS	13H	Make continuation line
3	MP	12H	Make new line
4	{		
5	}		
6		60H	
7		0H	
8	XYR	15H	Send x, y
9		10H	
A	Space		
B		10H	
C	~		
D	CCR	14H	Send character at cursor

10NOV81-V0-CRS

NEWBRAIN-TECH-8.1

01.00

8.1 INTRODUCTION

NEWIO is a collection of 4 drivers:-

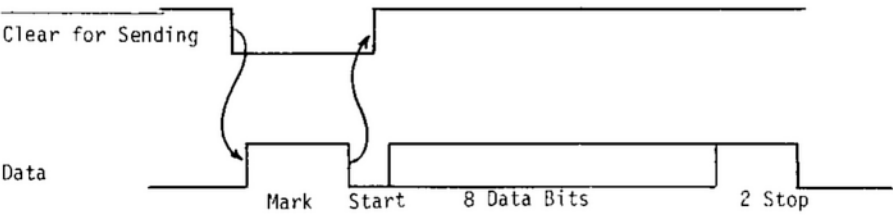
- LPIO, the line printer driver
- JGIO, general purpose bi-directional V24 driver
- UPIO, user port driver
- DUMMY, dummy interface to IOS.

The first two operate similarly and utilise the same code. Both expect the port number passed by IOS in the D register to be zero. Both use bits in the Enable Register and User Port of Model A type Newbrains to effect data transfer.

Signal	Location	Bit	Use
Clear to Send Printer	User Port	7	Input from printer indicating when it goes low that printer is ready to receive a data byte.
Transmit Data Printer	Enable Register	7	Serial data output to printer.
Clear to Send V24	User Port	1	Input from V24 device indicating when it goes low that device is ready to receive a data byte.

Signal	Location	Bit	Use
Clear for Sending	Enable Register	4	Output to V24 device indicating that Newbrain is ready to receive a data byte. The immediate response from the device should be to set input stream to Mark state if not already.
Received Data V24	User Port	0	Serial data input from V24 device.
Transmit Data V24	Enable Register	5	Serial data output to V24 device.

The serial data stream passing between the Newbrain and an external V24 type device is as follows:-



8.2 LPIO

The line printer driver is an output only device. No distinction is made between OPENIN and OPENOUT, either effectively performs an OPENOUT. Request code 2, INPUT, is not supported and if INPUT is attempted via IOS will result in an Illegal Request Code, Error 109.

8.2.1 OPEN - LPOPOU

The subroutine SOPN is called which analyses the open parameter string, calls MKBUFF and sets up own memory. The parameter string format for the line printer driver is:-

[T [tttt]] [R[rrrr]]

where T defines the start of the optional transmit field.

tttt is the transmit baud rate
which is optional but must follow T.

R defines the start of the optional receive field.

rrrr is the receive baud rate
which is optional but must follow R.

The parameter string is common for all software driven V24 devices. For the line printer only the transmit rate is used, the receive rate, should it exist, is ignored. The baud rate of the string tttt and rrrr are the actual baud rates of the attached devices, e.g. 2400, with a current design maximum of 19200 baud.

10 NOV81-V0-CRS

NEWBRAIN-TECH-8.2 02.00

Prior to the analysis of this string SOPN checks that the port number passed in D is zero. If not, error code 121 is set in the accumulator - Illegal Port Number, and an immediate return is made to IOS with carry set.

The length of the parameter string is checked not to be greater than 255, but if it is, error code 120, Baud Rate Parameter String Syntax Error, is set in the accumulator and a return made to IOS. The first character of the string is converted to upper case via HLTTCAPS, section 2.10 of OS. If the first character is a "T" then the next characters are analysed and if these are numeric characters, converted to a sixteen bit binary integer. The result is divided into the maximum baud rate, 19200, to derive the baud rate parameter (brp) as a fraction of the maximum rate. E.g. if the actual device baud rate is 2400 which when divided into 19200, gives 8, 8 is the brp parameter. If no numeric characters follow the T, or if no transmit field exists at all, then the default brp is 2, corresponding to a baud rate of 9600. The receive parameter is analysed similarly.

BC is set to 4 and MKBUFF called. The four bytes of own memory are set up as follows:-

Offset	Value
0	Receive Baud Rate Parameter
1	Print Line Length set to 80
2	Print Head Position initially set to 0
3	Transmit Baud Rate Parameter

On return from SOPN the last written byte to the Enable Register is retrieved from ENREGMAP. The IOPOWER Bit 3 and the Transmit Data to Printer Bit 7 are set into the byte, the result output to the Enable Register port with interrupts disabled. The registers B and C are cleared defining a zero length returned parameter string.

10NOV81-VØ-CRS

NEWBRAIN-TECH-8.2 03.00

8.2.2 OUTPUT - LPOUT

On entry HL points at start of own memory and this is incremented to retrieve the print line length. The output character in the accumulator is compared with the space character to determine whether it is a control character.

If it is not a control character then the current position of the print head, offset 2 of own memory is compared with the line length. If they are the same, a Carriage Return and a Line Feed character pair are output to the printer via LPWRT. The output character passed to the driver in the accumulator is then output by calling LPWRT. Unless any errors were detected by LPWRT, the Print Head Position parameter is incremented prior to a return to IOS.

If the character in the accumulator is a control character then one of the following actions is performed:-

- if it is a TAB character then eight spaces are output bringing the print head to a modulo 8 character position with a CRLF pair if the line length is exceeded.
- if it is a Carriage Return this is output followed by Line Feed.
- any other control character is passed directly to the printer.

8.2.2.1 LPWRT

BRKTST is called, section 2.7, which returns with carry set if the STOP key has been depressed; LPWRT returns immediately in this case. With interrupts disabled the User Port is input and the Clear to Send bit tested. The routine loops waiting for this bit to go low.

-

10NOV81-VQ-CRS

NEWBRAIN-TECH-8.2 04.00

The video, and thence the Bus Requests, is switched off via a call to NOVID in the Frame Interrupt Handler which resets the TVENA Bit 2 of the Enable Register. This is done so that the software delay loops in the following code of LPWRT generate accurate time intervals. With B set to 11 (1 start, 8 data and 2 stop bits), C containing the output character and carry clear the outermost loop is entered.

The data is passed to the output device via a bit within the Enable Register. The value of this bit is determined by the state of the carry flag on entry to the output loop. With carry set the bit is set, with carry clear the bit is clear. On first entry to the loop carry is cleared so that a start bit is output. With the output bit of the Enable Register set appropriately, the Enable Register port is written to. The routine then enters a delay loop, the number of iterations around which is determined by the transmit brp of own memory. E.g. if the brp = 1 equivalent to 19200 baud, the delay loop is not entered; if the brp = 2 the loop is performed once.

A further delay is implemented and then the C register is rotated right one bit so that the next send bit is set in the carry while a one bit is set in bit 7 of C.

The loop is executed B (11) times, the ones fed into C are the stop bits once the 8 character bits have been output. The video is re-enabled by setting bit 4 of TV2 which blanks TV after next Frame Interrupt but re-enables video by the interrupt after that. Interrupts are enabled, having been disabled for the duration of the character transfer and with HL pointing at offset 2 of own memory and carry clear the routine returns to caller.

8.2.3 CLOSE - UPOC

The registers A, B and C are cleared.

10NOV81-V0-CRS

NEWBRAIN-TECH-8.3 01.00

8.3 JGIO

This is a general purpose software driven V24 driver which operates through the Enable Register for output and the User Port for input. No distinction is made between OPENIN and OPENOUT, the same routine being called for each. Both INPUT and OUTPUT are supported.

8.3.1 OPEN - JGOP

Behaves similarly to LPOPOU section 8.2.1. SOPN is called which analyses the parameter string, calls MKBUFF and sets up own memory. The IOPOWER Bit 3 and Transmit Data to V24 Bit 5 are set in the last written byte to the Enable Register which is then output to that port. B and C are cleared defining a zero length returned parameter string.

8.3.2 INPUT - JGIN

The TVENA and Clear for Sending bits of the Enable Register last written byte are reset and the resulting byte output to the Enable Register port. If not already at the Mark state, the input bit from the V24 device should go high in response to Clear for Sending going low. The routine waits in a loop until this bit, Bit 0 of the User Port, becomes high. DWAIT is called which waits for the input stream to go low indicating the start bit. This routine can wait for up to 1 minute after which a timeout error 200 is set in the accumulator and with carry set returns to JGIN's caller normally IOS.

Once the stop bit is achieved, the Clear for Sending bit of the Enable Register is set high. A delay loop is executed the Receive brp times. The input data is read through Bit 0 of the User port and the result shifted into the

carry and thence into the E register where the input character is compiled. A fixed delay of 12 DJNZ instructions and a delay based on the Receive brp are performed so that on completion the next input bit from the V24 device will be ready at the User Port.

The loop ends when the start bit is shifted through the E register into the carry, at which time the E register will contain the eight data bits. Video is re-enabled as described in section 8.2.2.1 and interrupts enabled. Carry is cleared and the character in E is set in the accumulator for return to IOS.

8.3.3 OUTPUT - JGOUT

Sets the V24 mark bits in D and E positions, the own memory pointer in HL at offset 2 and executes LPWRT, section 8.2.2.1.

8.3.4 CLOSE- UPOC

The registers A, B and C are cleared. Carry is not set.

8.4 UPIO

The user port driver supports request codes 0-4 and performs the following:-

- OPENOUT or OPENIN, clears A, B, C and Carry.
- INPUT, inputs the 8 bit wide data from the port number passed in D from IOS to the accumulator.
- OUTPUT, outputs the data in the accumulator to the port number specified in D.
- CLOSE, clears A, B, C and Carry.

10NOV81-V0-CRSNEWBRAIN-TECH-8.5 01.00

8.5 DUMMY

Does nothing but provide a clear interface to IOS for request codes 0-4,
for each code merely clears A, B, C and Carry.

13NOV81-V0-CRSNEWBRAIN-TECH-9.1 01.00

9.1 INTRODUCTION

Newbrain BASIC is an interactive compiler. Statements are entered from the console and as each line is entered a minimal lexical check is performed with BASIC reserved words converted into single byte representation. The compressed statement line is added to the Source Code area, its line number, set in the Line Number area which is maintained in numeric sequence. At execution time, the source code is conditionally compiled into an intermediate object code saved in the Object Code area. The object code provides a set of pointers into a table of execute routines. As each statement is executed, the compilation phase is only performed if the object code does not exist.

BASIC is an example of a user program which exists in high RAM but addresses the full random access memory through accesses to page zero variables, stack usage, the floating point memory through calls to the Maths Pack, and IOS memory when it performs IO. It calls OS, IOS and Maths Pack routines via the intercept mechanism, picking up routine addresses from ZOSTABLE. Its own internal routines are called directly or via the intercepts through RST16 and RST24 which call BASIC routines through ZTABLE.

The RAM organisation is:-

low address		Section	
B3PRM	Fixed Locations	IY	9.1
	Source Code Area	IY + 61	9.2.1
	Object Code Area	STOP	9.2.3
	Free Memory	QTOP	
	User Stack	USRSTP	9.2.4.6
	FOR Block area	FORPTR	9.2.4.5
	GOSUB Stack	GSSPTR	9.2.4.4
	Strings	STBAS	9.2.4.3
	Arrays	ARBAS	9.2.4.2
	Symbol Table	SBAS	9.2.4.1
	Line-Number Table	LBAS	9.2.1
	High address		

13NOV81-V0-CRS

NEWBRAIN-TECH-9.1

03.00

The fixed locations are:-

Displacement from IY	Length	Name	Value
0	6		Unused
6	1	OUTCON	The IOS stream to which output is currently being made.
7	1	INCON	The IOS stream from which input is currently being taken. GET and PUT use displacements 6 & 7 in a special way. Stream 0 is used by BASIC as the console.
8	1	ERRNO	The number of the last error or interrupt to have occurred in program execution.
9	2	INPTR	The position from which RDCH and other input routines are taking input from stored text.
11	1		Unused.
12	2	SVINPT	A location for saving INPTR.
14	2	STOP	Points to top of source code.
16	2	OTOP	Points to top of object code.
18	2	USRSTP	Base of user stack.
20	2	FORPTR	Base of the FOR blocks area.
22	2	GSSPTR	Base of the GOSUB stack.
24	2	STBAS	Base of the String area.

13NOV81-V0-CRS

NEWBRAIN-TECH-9.1

04.00

Displacement from IY	Length	Name	Value
26	2	ARBAS	Base of the Array area
28	2	SBAS	Base of the Symbol Table
30	2	LBAS	Base of the Line Number Table
32	2	TOP	Top of the BASIC RAM area which can be moved down with the RESERVE command
34	1	FLAGS	<p>Bit 0 set during execution used by INITOB inhibits deletion of current object code line.</p> <p>Bit 1 set if interrupts are to be ignored</p> <p>Bit 2 set if tokens are not to be expanded during listing (e.g. during SAVE)</p> <p>Bit 3 set during compilation used by INITOB inhibits deletion of object code line.</p> <p>Bit 4 is set whilst compiling a DEF so VARIAB can watch for formal parameter occurrences</p> <p>Bit 5 is set if compilation is not to produce object code</p>
35	2	NCONNO	Count of number of numeric constants encountered to give each one a unique name type. Not currently implemented.
37	2	SCONNO	Count of number of string constants encountered. Bit 14 is always set. Not currently implemented.

13NOV81-V0-CRS

NEWBRAIN-TECH-9.1 05.00

Displacement from IY	Length	Name	Value
39	3	CLINNO	Holds the line number and displacement where last break of execution occurred.
42	1	BASFLG	Bit 0 = base of arrays either 0 or 1 Bit 7 = lower base is now fixed if set Set by DIM statement or OPTION BASE statement.
43	1	OTYPE	Temporary variable for EXPN, the expression compiler. Set to 0 if a numeric expression found or 10H if string expression.
44	2	OMOVE	The amount by which the object code has moved during execution of a line always negative usually zero.
46	1	PPFLGS	Used by AMENDL and XLIST, indicates state during entokening source line.
47	2	DTINPT	Like INPTR but used for reading DATA statements. Only one byte is used.
49	2	DTLN	The current line number that the DATA pointer is on. Changes to the Line Number table entry during execution of a READ.
51	2	FORVAR	Temporary variable during compilation of a FOR statement.
53	2	APAR	Absolute pointer into the user stack of the actual parameter being evaluated in a User Defined Function.
55	2	FPAR	Holds the name type of the parameter being evaluated in a User Defined Function.

13NOV81-VO-CRSNEWBRAIN-TECH-9.1 06.00

Displacement from IY	Length	Name	Value
57	2	ERRLIN	Where control is passed to if program error. If no trap exists then it is zeroised.
59	2	BRKLIN	Where control is passed if a break occurs. If no trap exists then it is zeroised.

9.2 BASIC Main Loop

The entry point to BASIC is the label BASIC and is the default value set in the page zero location UP by OS during the power up sequence, section 2.2. The BASIC program initialises itself and opens the console and uses DEV0 as device number, DPSP and DPSL as the parameter string and zero as the port number and then enters a loop which will either be a statement or a command, the latter indicated by the absence of a line number.

9.2.1 Statements

The routine AMENDL performs the initial processing of statements. Each BASIC statement can be considered composed of:-

- a line number
- an initial keyword and parameters
- other keywords and parameters.

The keywords are each converted into single byte tokens. A cross reference list of keywords and tokens is held in the table KWDLIS. The single byte tokens have bit 7 set and therefore are in the range 128 to 255. There are two sets of tokens; one set refers to initial keywords, the second set to other keywords. Tokens in the two sets may have the same value but a different meaning. For example, token value 128 is equivalent to LET as an initial keyword and NOT if it is another keyword. All keywords, functions, operators and BASIC connective words (THEN, TO, STEP, etc) have a corresponding token. Some have two. ? and PRINT mean the same thing but are assigned two different tokens so that on conversion back to the full BASIC form when LISTING the program, the statements are reproduced as entered. AMENDL also performs a minimal syntax check specifically to identify IF THEN statements. Newbrain BASIC does not normally require the THEN to be inserted e.g. IF X=Y LET A=B. The BASIC compiler however cannot handle implied THENs so AMENDL inserts an "invisible" THEN token at the

correct location within the statement.

The entokened statement, which is shorter than the input source statement, is added to the top of the Source Code Area, STOP. All source lines of the program and any command line are held here. When old lines are deleted the others are moved down. Each one ends in a Carriage Return which replaces the : in multi-statement lines which are split in the Source Area. The ordering of the source lines is not significant, program statement sequence being maintained in the Line Number table. The statement line number is inserted into the Line Number table in correct numeric sequence. A line number table entry has the format:-

Offset	Length	Contents
0	2	The line number
2	2	Pointer to first character of a statement in the Source Code Area
4	2	Pointer to the Object Code

At this stage the Object Code pointer does not exist and is set to zero. The entries are always held in line number order, low numbered lines having low addresses. Multi-statement lines have a separate entry for each statement, each having the same line number, and are held in the correct textual order.

9.2.2 Commands

If the input string from the keyboard is interpreted as a command it is handled by the routine `COMMAN`. This assigns a line number of 0 and calls `AMENDL` which entokens the line and sets an entry, or entries, at the low address end of the Line Number table.

9.2.3 Compilation

BASIC executes a program by calling EXEC. This operates by inspecting the Line Number table. EXEC can determine the sequence that statements are to be executed and the address of the object code. On first pass through, the object code pointers will be zero indicating that the corresponding source statement has not been compiled. EXEC calls CMOBJ which compiles a statement.

The tokenised statement in the Source is analysed, the syntactic position of each token determines whether it is an initial keyword token or another keyword token. For each initial token there is a corresponding Compile module, the module name prefixed with the letter C. Other syntactic objects may have corresponding C modules e.g. CLABEL is the compile routine for syntactic object line number. The appropriate C routine for a token is effected by the routine STATEL. This is called via ZTABLE and may therefore be intercepted, see section 2.6.

The output from the compile modules for a statement is an object code record composed of Y-codes and parameters. Y-codes are one byte operators, which currently may have 0, 1 or 2 parameters. At execution time the compiled sequence of Y-codes results in a sequence of calls to a corresponding set of execute routines prefixed with the letter X. The Y-code is an index into the table XTABLE which provides the address of the corresponding X routine.

The compiled object code line is set at the top of the Object Code Area, OTOP. Each line is preceded by a two byte count giving the total length of the object code record. The address of the first byte of object code after the count is set in the Line Number table.

9.2.4 Execution

EXEC extracts the object code pointer from a Line Number table entry. The X routines are determined and executed by XCL3. Normally statements are only compiled once on first pass through the program with the result that execution is very efficient. Occasionally, for example when free memory becomes exhausted, the object code is deleted, and the object code pointers in the Line Number table set to zero. As subsequent statements are executed they will require to be compiled. No significant overhead is caused by this unless there is very little free memory. Since the object code consists of pointers into both the Symbol and Line Number tables which are fixed during execution, instructions that alter them (e.g. DELETE, RESERVE, CLEAR, MERGE) also cause the object code to be deleted. IX is used during execution as the Line Number Table pointer and behaves like a program counter.

At execution time a number of tables are used:-

- Symbol Table
- Array Area
- String Area
- GOSUB Stack
- FOR block area
- User Stack

These tables expand towards lower memory addresses with for any particular entry offset 0 at the low memory end of the entry. The RUN command initialises all tables so that all base addresses are concurrent with LBAS.

9.2.4.1 Symbol Table

Symbols and constants are added to the base of the table (SBAS) as they are encountered which preserves the validity of the absolute pointers in the object code. Deletion of symbols is by the CLEAR statement. Entries may be selectively removed by CLEAR, remaining entries are moved up destroying the integrity of object code pointers, necessitating the deletion of the object code. All symbols are 8 bytes in length. The first two bytes contain the name type. For a variable, the low order 10 bits is a coding of the variable, which is:-

$$37* (\text{First letter}) + C$$

where First letter is the first letter of the variable mapped into the range 0-25 e.g. A=0.
 C=36 if no second character
 or C=26 + digit if digit is the second character
 or C is the second letter of the variable mapped into the range 0-25
 e.g. variable B7 is coded as
 $37*1 + 26+7$

For a numeric constant the name type is 0H, for a string constant 4000H and the system variable FILE\$, which contains the last retrieved parameter string from IOS, has a special name type that cannot occur for constants or variables.

The remaining bits of the name type bytes are used as follows:-

Bit	Value
10	Unused
11	1 = user defined function, 0 otherwise
12	1 = 1 dimensional array or if a user defined function, bit 11 set, then function has a numeric parameter
13	1 = 2 dimensional array or if a user defined function bit 11 set, then function has a string parameter
14	1 = string constant, string array, string variable or if a user defined function with a string result
15	0 = constant, 1 otherwise.

For an array, if both bits 12 and 13 are set, this means that it is not certain yet what number of dimensions the array has. For a user defined function if neither bits 12 and 13 are set the function has no parameters.

The other 6 bytes are used as follows:-

- numeric variable or constant, the floating point value of the constant or current value of the variable.
- string variable or constant

Offset	Contents
2	Position in String Area relative to ARBAS
4	Length of String
6	Unused

- arrays

Offset	Contents
2	Position of Array relative to SBAS
4	1st Dimension
6	2nd Dimension, or 0 if 1 dimensional array

- user defined function, the six bytes are unused.

13NOV81-VO-CRS

NEWBRAIN-TECH-9.2 07.00

9.2.4.2 Array Area

Contains the elements of arrays. The start address of the first element of an array is contained in the Symbol Table entry for the array name. The Symbol Table entry specifies the array start as an offset from SBAS since SBAS may move as symbols are added or deleted. When array are deleted by CLEARing the array name the offsets of all the arrays in the symbol table are updated.

The elements of an array are held as six byte entries in the Array Area. For numeric arrays the six bytes contain the current floating point value of the array element. For string arrays the six bytes are used in the same way as the last six bytes in the Symbol Table.

Offset	Contents
0	Position in String Area relative to ARBAS
2	Length of String
4	Unused

Low numbered elements live in low memory addresses. For two dimensional arrays the elements are ordered (0, 0), (0, 1), ..., (1, 0), (1, 1), (1,2) etc. The offset of an element relative to offset 0 of the array is:-

$$((1st\ subscript) * (2nd\ Dimension) + (2nd\ subscript)) * 6$$

9.2.4.3 String Area

The String Area contains only the actual strings which are in no particular order. As strings are modified, extended or shortened, they are repositioned to the base of the string array area STBAS, with the vacated space closed up. Consequently, string pointers in the Symbol Table and Array Area have to be updated. Pointers into the String Area are relative to ARBAS as the contents of the Symbol Table and Array Area may change.

13NOV81-VO-CRSNEWBRAIN-TECH-9.2 09.00

9.2.4.4 GOSUB Stack

This consists of 3 byte entries containing the Line Number table entry addresses of the GOSUB statements that have yet to be RETURNed from, the most recent are at the lower addresses. Three bytes are used so that when in command mode the current position in a multi-statement line may be determined absolutely. In these circumstances the first two bytes are the actual line number and the third byte is the statement number in the current line with first statement being 1.

9.2.4.5 FOR Block Area

A FOR Block is twenty bytes of the following format:

Offset	Length	Value
0	3	Line Number table entry of NEXT statement
3	3	Line Number table entry of FOR statement
6	6	Floating point value of STEP factor
12	6	Floating point value of TO limit
18	2	Symbol Table entry of FOR variable

The line number entries use three bytes as for the GOSUB stack.

13NOV81-VO-CRSNEWBRAIN-TECH-9.2 11.00

9.2.4.6 User Stack

This stack is used by the object code interpreter when a Y-code demands a push or pop. FOR blocks grow initially out of the user stack and get taken over by FORPTR. The stack is used for passing actual parameters and link information in User Defined Function execution. The stack is also used as the RDLIN input buffer. At times INPTR and SVINPT, when amendments are taking place in lower memory, and if pointing to an input buffer or the user stack, then these will be relative to the user stack rather than absolute addresses.

9.3 META-MODULES

The groups of modules which support the BASIC interpreter operation are:

- Compile Routines
- Atomic Compilation Routines
- Symbol Table module
- X Routines
- Input Stream module
- Output Stream module
- Working Store Manipulations

9.3.1 Compile Routines

The source statement token is used by STATEL to extract the corresponding C routine. Each initial keyword token has a corresponding compile routine e.g. TGOTO corresponds to CGOTO. Other keyword tokens do not always have a corresponding C routine whilst certain syntactic structures, such as the line number argument following a GOTO, have a C routine in this case CLABEL. Included in these modules is the expression compiler.

The statement `LET C = A + B * 2` would be compiled into the following:-

```
YPUSH2 <symbol table address C >
YPUSH2 <symbol table address A > YDEREF
YPUSH2 <symbol table address B > YDEREF
YNCONST <symbol table address "2" > YTIMES YPLUS YLETN
```

Variables are compiled using VARIAB and may require an addition to the Symbol Table via the Symbol Table metamodule.

9.3.2 Atomic Compilation Routines

As object Y-codes are created they are passed through CMONE to be put into the object code area. This is interceptable through OS if the object code is to be stored somewhere else. Includes the useful routines:-

- CREM, compiles YREM which at execute time is a No-op
- CPSHØ, compiles YPUSH2, Ø, Ø, puts two zeroes on the user stack at execution time
- CPSH, compiles YPUSH2 which push the following two bytes in the object code record onto the user stack
- CMTHRE, compiles codes in order A, L, H where A might contain a Y code and HL a pointer(A,L,H are Z8Ø registers)
- CM2HL, compiles codes in order L, H
- CMONE, compiles codes in A which might be a Y-code or argument.

A flag may be set to inhibit the generation of object code, Bits of IY+FLAGS

9.3.3 Symbol Table Modules

Performs the creation and deletion of symbols for arrays, string, variables, constants and user defined functions.

9.3.4 X Routines

Called by corresponding Y-code performs stack operations on strings numbers and arrays.

9.3.5 Input Stream Module

Provides BASIC with an internal read from stored text either read into a buffer via OS or internally. RDLIN is the main input routine and reads a line of input from stream E onto the user stack. The input line is always expected to end with a Carriage Return.

9.3.6 Output Stream Module

Outputs direct to the current output stream in OUTCON. Included are routines to:

- PTCH, write a character
- PTMSG, write a message. This is used to output an in-code message and is used as:-
 CALL PTMSG
 DEFB <byte count>
 DEFM "<message>"
- PTNL, write NEWLINE, outputs a single 0DH character
- PTEDF, write End of File mark, two characters 04H, 0DH

9.3.7 Working Store-Manipulations

Manages memory storage for BASIC:-

- adding and deleting entries from tables
- MKSPC and RETSPC routines.

-

10.1 INTRODUCTION

Floating point numbers are stored as 6 bytes with an internal range of $\pm 10^{\pm 150}$. This internal format has a precision in excess of 10 significant digits and all calculations are carried out to the full precision. The floating point algorithms have been designed to minimise cumulative rounding errors and maintain stability in interative calculation. Numbers output in ASCII from the maths pack may only be in the range $\pm 10^{\pm 99}$. Output less than 10^{-99} is set to zero, output greater than 10^{+99} results in an overflow error. This error is not signalled in the carry flag - instead ****'s are output in the appropriate field width. In formats other than free format underflow can be an error and's are output. All other errors detected cause the carry flag to be set on return but no error number is set in the Accumulator.

The Maths Pack has 160 bytes of memory reserved for its use starting at address 100H. The first six bytes of this memory is the floating point accumulator FACC which is used to pass operands between the calling user program and the Maths Pack. Although the entire Maths Pack memory may be used by any particular floating point routine, between calls only locations 137H to 13AH (which contain the random number seed) must be preserved if being used. The remaining memory is available for use and so in theory the stack area may grow down into the Maths Pack memory beyond the allocated memory provided these rules for avoiding conflicts are adhered to.

10.2 ROUTINES

Maths Pack Routines can be categorised into:-

- zero argument routines
- single argument routines
- double argument routines
- special routines

10.2.1. Zero Argument Routines

Exit: Answer in FACC
 HL points to FACC
 ABCDE destroyed
 Carry always clear

Functions: PI the trigonometrical constant π
 FPONE the floating point number
 FPZERO the floating point number 0
 FPMONE the floating point number -1

RND a pseudo random number from the uniform
 distribution 0-1

STPOS returns "PEEK" of print head position from page zero
 location PHPOS

16 NOV81-V0-CRS

NEWBRAIN-TECH-10.2

02.00

10.2.2 Single Argument Routines

Entry: Argument in FACC

Exit: Answer in FACC
 HL points to FACC
 Carry set only if an error
 ABCDE destroyed

Functions: NOT Boolean operation - numbers are truncated to integers in the range -32, 768 to +32, 767. If this cannot be done an error will be returned. A bitwise binary operation is then performed and the result floated to provide the the answer.

NEG Negate

ABS Absolute value

INT Integer part (greatest integer \leq)

SIGN Sign (returns - 1, 0 or 1)

SQRT Square root

SIN Trigonometric function

COS Trigonometric function

TAN Trigonometric function

ASIN Inverse Trigonometric Function($-\pi/2, \pi/2$)ACOS Inverse Trigonometric Function (0, π)ATAN Inverse Trigonometric Function ($-\pi/2, \pi/2$)

LOG Natural logarithm

EXP Exponential functions

NOP Unary Plus

PEEK Fixes the address in FACC and access the resulting 16 bit address. The byte recovered is floated into the Floating Point Accumulator.

10.2.3 Double Argument Routines

Entry: Arg 1 in FACC
DE points to low address of Arg 2

Exit: Answer in FACC
HL points to FACC
Carry set if error
ABCDE destroyed

Functions: AND see NOT section 10.2.2

OR ditto

MULT Multiply

DIV Divide Arg 1 - Arg 2

ADD Add

SUB Subtract Arg 1 - Arg 2

RAISE Exponentiation (Arg 1) Arg 2

NUMGE Greater than or equal

NUMNE Not equal

NUMGT Greater than

NUMLT Less than

NUMEQ Equal

NUMLE Less than or equal

} Compares Arg 1 with Arg 2

Result is -1 if true

Ø if false

e.g. NUMLE gives true if

FACC \leq (DE)

16NOV81-V0-CRSNEWBRAIN-TECH-10.2 04.00

10.2.4. Special Routines

10.2.4.1 LDF Load the Floating Point Accumulator

Entry: HL points to operand

Exit: HLBC preserved
 ADE destroyed

10.2.4.2 STF Store the Floating Point Accumulator

Entry: HL points to destination

Exit: HLBC preserved
 ADE destroyed

10.2.4.3 FIX Convert to 16 Bit Positive Binary

Entry: HL points to operand

Exit: DE = answer
 ABCHL destroyed
 FPACC destroyed
 Carry set if overflow

10.2.4.4 FLT Convert 16 Bit Positive Binary Floating Point

Entry: DE = operand

Exit: FACC = answer
 HL points to FACC
 ABCDE destroyed

16NOV81-V0-CRS

NEWBRAIN-TECH-10.2 05.00

10.2.4.5 COMP Compare

Entry: HL points to first operand
DE points to second operand

Exit: Carry is set if first operand greater than second clear
otherwise
Zero flag set if operands are equal

10.2.4.6 INP Convert ASCII to Floating Point

Entry: DE points to first character of string

Exit: FACC = answer
HL points to FACC
DE points to first unused character in string
ABC destroyed
Carry set if input number too large

[+ | -] (digit* [. [digit] * |.digit*) [E [+|-] digit [digit]]

where [] indicates an optional field

() indicates mandatory field

| indicates "or"

digit is ASCII character in range "0" to "9"

* indicates may be repeated

E is ASCII character "E" or "e"

Spaces are disregarded

The + or - characters may be ASCII or the NewBrain BASIC
token + or -.

16NOV81-V0-CRS

NEWBRAIN-TECH-10.2 06.00

10.2.4.7 OUT, Convert Floating Point to ASCII

Entry: Number in FACC
BC = format code

Exit: HL points to output string
C = number of characters in string
ABCDE destroyed
Carry always clear

Format Coding:

bits 6,7 of B = format code, f
bits 0-5 of B = integer digit count, i
C = total digit count, t

f = 0 : Fixed point format
Number is rounded to decimal with t
digits, t-i decimal places. Leading
zeroes represented as spaces, 1 leading
space or minus sign, 1 trailing space.
Total field width t + 3, except when
t = i (integer format), t + 2.

f = 1 : Free format
Number is rounded to t significant figures and output
in fixed or exponential format - goes to
exponential above 10^t or below 10^{-3} .
Underflows are converted to ASCII 0.
The field width is variable.

f = 2: Exponential format
 Number is output as fixed point format with two digit
 exponent a multiple of i field with t + 7.
 For all formats must have $10 \geq t > i > 0$.

The answer is returned as an ASCII string in the Maths Pack memory area. Maximum string length is 17 characters. The string may become invalid after subsequent calls to routines which use the Maths Pack memory area. No errors are returned but attempting to fit number into inappropriate formats will result in the output string containing asterisks on overflow or periods on underflow.

10.2.4.8 RANDOM

Function: The "RANDOMIZE" function seeds the random number generator the value of page zero location CLOCK.

10.2.4.9 INITRAND

Function: Seeds the random number generator with a standard value.

10.2.4.10 ROUND 16 Bit Positive Binary of (FACC) + 0.5 (as FIX)

Entry: HL points to operand

Exit: DE = answer
 ABCHL destroyed
 Carry set if overflow

24NOV81-V0-CRS

NEWBRAIN-TECH-10.2

08.00

10.2.4.11 MATHS, interface to 0, 1 and 2 argument maths routines.

Entry: A = 1 byte routine code
 HL points to 0, 6 byte or 12 bytes data area.

The Maths Pack routines may be called through the general interface routine MATHS. This picks up entry addresses from the OS intercept table ZOSTABLE. Three sub-tables exist within ZOSTABLE.

- AR0TAB, the list of 0 argument routines
- AR1TAB, the list of 1 argument routines
- AR2TAB, the list of 2 argument routines

The special floating point routines are not accessible through this interface. The Accumulator contains an entry code, bits 7 and 6 determine the sub-table, bits 5 to 0 determine the offset of the required routine within the sub-table. The bits 7 and 6 selection are:

- neither set selects AR0TAB
- bit 7 set selects AR1TAB
- bit 6 set selects AR2TAB

HL on entry points to a data area which contains 1 or 2 operand data.

A.1 HARDWARE PORT ALLOCATION

Port 0, CLUSR, Clear User Interrupt, read/write.

- No data transfer is associated with port 0. A read or write causes the interface signal CLUSR, Clear User, to be strobed. The interface signal USRINT should be reset.

Port 1, ENRG2, Load 2nd Enable Register, write

- The 2nd Enable Register is not currently defined.

Port 2, PR, Load Page Register, write

- Available in Model M machine only. Causes an 8k memory module to be paged into the Z80 address space. The page address and memory module are specified as follows:-

Bits	Value
0-6	Memory module identifier
7	ROM/RAM identifier.

Additionally A13, A14, A15 of the address bus are set to the Z80 address space page number. A8-A12 are undefined.

Port 3, USER, Load User Output Port, write

- A byte of data is written to the user port latch. The signal USTRB is strobed.

Port 4, CLCLK, Clear Clock Interrupt, read/write

- No data transfer is associated with port 4. A read or write causes the CLCLK to be strobed which clears the clock interrupt. Interrupts only reach the Z80 if the corresponding enable is set. In this case the CLCLK signal need only be activated if the clock enable has been set via the 1st enable register.

Port 5, CLDMA, Clear DMA Interrupt, read/write

- Not currently implemented but will behave similarly to the port 4 Clear Clock Interrupt.

Port 6, COP, Address COP's chip, read/write

- The COP I/O Port.

Port 7, ENRGT, Load 1st Enable Register, write

- A data byte is written to the 1st Enable Register. Two principle byte formats exist. For the Model A this is:-

Bit	Value
0	<u>Clock Enable</u>
1	Not Used
2	TV Enable
3	Not Used
4	<u>Clear for Sending</u> V24
5	Transmit Data V24
6	Not Used
7	Transmit Data Printer

For Newbrain the Enable Register format is:-

Bit	Value
0	Clock Enable, enable clock interrupts
1	User Enable, enable user interrupts
2	TV Enable, enable video
3	V24 Enable, synonymous with V3 power enabled
4	V24 Select Receive Bit 0
5	V24 Select Receive Bit 1
6	V24 Select Transmit Bit 0
7	V24 Select Transmit Bit 1

The Bits 5 and 4 select one of four V24 inputs as follows:-

Bits 5, 4	Selection
00	Tape Loop 0
01	Tape Loop 1
10	Unassigned
11	Serial Communications

Similarly Bits 7 and 6 select one of four V24 outputs as follows:-

Bits 7, 6	Selection
00	Tape Loop 0
01	Tape Loop 1
10	Printer
11	Serial Communications

For Model V, bits 0 and 2 of the 1st Enable Register are not used since there is no TV and therefore no clock requirement. Bit 3 is not used as V24 is permanently disabled. Also for Model V only the Printer and Serial Communications of the four possible V24 directions reach an I/O connector.

Additionally, for all models, bits 5 and 4 of the 1st Enable Register select one of four directions for Bit 0 of the Status Register, port 20; bits 7 and 6 likewise select one of four directions for Bit 1 of the Status Register.

Ports 8 & 9, \overline{TVL} , Load TV Addresses, write

- The TV RAM character buffer starts on a 64 byte boundary in the memory address range 0-32k. Under these conditions this start address may be specified in 9 bits. The address bits A7-A14 are specified through port 9. No data is transferred through port 8 but, the act of writing to port 8 causes the bit 6 of the TV RAM start address to be set. The load TV address is performed each frame, the hardware resets bit 6 of the start address at frame end.

Ports 10 & 11, are undecoded and appear as ports 8 & 9.

Port 12, \overline{TVCTL} , Load TV Control Register, write

- A data byte is written to the TV Control Register and controls the access and display of data from the character ROM. Two full 256 character sets are stored. The TV Control Register bit assignments are:-

Bit	Value
0	Reverse Video
1	1 = Full set of 256 characters 0 = first 128 characters with bit 7 of character address as an attribute
2	For model M specifies a 2nd set of 128 characters
3	UCR, upper character ROM, selects second character set and also selects 8TV lines per character (standard is 10) giving 30 character lines per screen.
4,5	Unused.
6	0 = 40 characters per line, 1 = 80 characters per line.
7	Unused.

Ports 13, 14 & 15 are undecoded and appear as port 12.

Ports 16, 17, 18 and 19 are reserved for Read Analogue Data.

Port 20, UST, Read Status, read

- A byte of data is read from the status register. Three types of information are returned:-

- 1) The interrupting device in the event of an interrupt
- 2) Hardware statuses
- 3) Odd input signals e.g. calling indicator, user status.

The condition of bits 0 and 1 are determined by the condition of bits 4-7 of a previously written Enable Register, port 7.

Bits	Value	Enable Register Bits 7 & 6
0	CALLIND, Calling Indicator	11
0	tape in, the digitised audio output from the cassette	10
0	1=40 characters per TV line 0=80 characters per TV line	01
0	1=excess of 24 0=excess of 4	00

Enable Register Bits 5 & 4

1	<u>CALLIND</u> , Calling Indicator	11
1	<u>BEE</u> , if set then processor is Model A type	10
1	<u>TVCNSL</u> , if set then processor has video device as primary console output.	01

Bits	Value
1	PWRUP if set indicates that power is supplied to Z80 and memory 00
2	Mains Present, indicates for battery backup models that, high consumption circuitry may be activated.
3	User Status
4	<u>User Int</u> , when set indicates an interrupt from the user device
5	<u>CTock Int</u> , when set indicates an interrupt from the 20mS video clock
6	<u>ACIA Int</u> , when set indicates an interrupt from the ACIA
7	<u>COP Int</u> , when set indicates an interrupt from the COP

Port 21, , User Input for Newbrain and Model V, read

- Reads a byte of data from the user port for the specified processor models.

Port 22, , User Input for Model A, read

- Supplies input signals for software driven minimal V24 interface and a route for tape in signal

Bit	Value
0	Received Data V24
1	<u>Clear to send V24</u>
2-4	Not Used
5	Tape in, the digitised audio output from cassette
6	Not Used
7	<u>Clear to Send Printer</u>

Port 23, undecoded and appears as port 21.

Port 24, ACIA Control/Status Registers, read/write

- A read from port 24 returns the 6850 status register.
- A write to port 24 loads the 6850 control register.

Port 25, , ACIA Transmit/Receive Registers, read/write

- A read from port 25 returns the 6850 data out
- A write to port 25 loads the 6850 data input register

Port 26, 27, Unused.

Port 28, , CTC Channel 0, read/write

- A write to port 28 loads the 280 CTC with a timing or counter value. Connected to the ACIA Receive Clock this port is usually used in counter mode
- A read from port 28 returns the counter constant.

Port 29, , CTC Channel 1, read/write

- As port 28 but connected to the ACIA Transmit Clock.

Port 30, , CTC Channel 2, read/write

- As port 28 but connected to clock input of CTC channels 0 and 1.

Port 31, , CTC Channel 3, read/write

- As port 28 but not used.

The clock input to CTC channel 2 is the system clock of 4MHz divided by 13 giving a clock rate of 307692Hz. With the CTC divisor set to ± 1 and ACIA divisor set to ± 16 via the control register the ACIA output clock frequency is as follows:-

CTC 0, 1 Divisor	ACIA Output
64	300.48 Hz
32	600.9 Hz
16	1201.9 Hz
8	2403.8 Hz
4	4807.7 Hz
2	9615.4 Hz

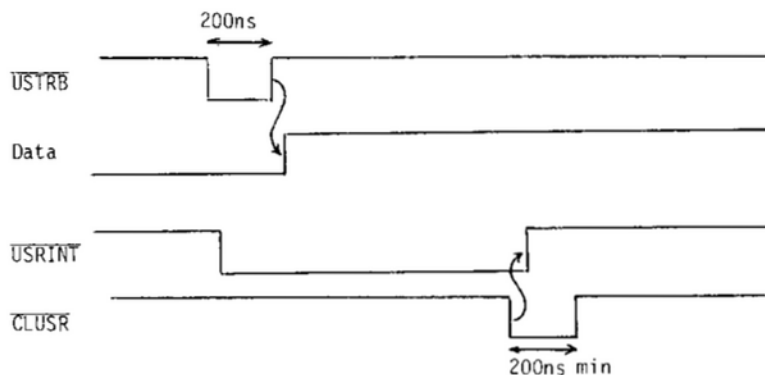
A.2 USER PORT INTERFACE

The User ports provide a simple bi-directional parallel interface whereby a user device may be connected to a Newbrain processor. Either or both of the input and output parts may be used. The signals are:-

	Pin No.	Function
<u>User Port In</u>	1	D0
	2	D1
	3	D2
	4	D3
	5	D4 Data into Newbrain
	6	D5
	7	D6
	8	D7
	9	USRST, User Status, into Newbrain
	10	USRINT, User Interrupt, into Newbrain
	11	CLUSR, Clear User, from Newbrain
	12	GND, ground

	Pin No.	Function
<u>User Port Out</u>	1	D0
	2	D1
	3	D2
	4	D3
	5	D4 Data from Newbrain
	6	D5
	7	D6
	8	D7
	9	USTRB, User Strobe, from Newbrain
	10	USRINT, User Interrupt, into Newbrain
	11	CLUSR, Clear User, from Newbrain
	12	GND, ground

The significant timing diagrams for the User Port interface are shown below.



Data from the Newbrain becomes valid on the rising edge of USTRB. To allow for propagation delays the data lines should be read 100ns after USTRB becomes high. The CLUSR signal is a request by the Newbrain to reset USRINT. USRINT should go high while CLUSR is low. Other signals and protocols can be defined by the user and their interpretation catered for within the user driver software.

APPENDIX B

LIST OF ROUTINES

ADHLA	6.4.1	GETCUR	6.4.7
ADLIN	6.4.4	GETCRØ	6.4.8
ADLIN1	6.4.5	GETDE	6.4.6
		GETKEY	2.4
BACAL	6.4.2	GETREG	5.3.2
BRKTST	2.7.1	GOTCHS	5.4.2
		GSPCBK	2.3
CALIN	6.4.3		
CALLNC	2.6	HLTTCAPS	2.10.1
CALLOS	2.6		
CALLRTN	2.6	ILIN	6.4.10
CASSERR	2.5.1.2	IMMKEY	2.4
CASSIN	2.5.1.3	INCDE	6.4.10
CASSOUT	2.5.1.5	INPUT(10S)	3.3
CLOSE(10S)	3.4	INTRPT	2.5
COPIS	2.5.1	IOC	3.5
COSNC	2.6	IOPOFF	2.9.2
CPDB	6.4.9	IOPON	2.9.1
CURLT	6.3.4.8		
CURRT	6.3.4.2.6	JGIN	8.3.2
		JGOP	8.3.1
FDE	3.6	JGOUT	8.3.3
FDEV	3.8		
FRINT	2.5.2		
FSTRM	3.7		

KBCLS	7.5	TLOPIN	5.2
KBD	2.5.1.4	TOS	= Top of Stack
KBIINP	7.3	TP10PI	4.4
KBOPI	7.2	TP20PI	4.4
KBOUT	7.4	TP10PO	4.2
KBWINP	7.3	TP20PO	4.2
KLOOK	7.7	TPCLS	4.6
		TPINP	4.5
LIOPIN	5.2	TPOUT	4.3
LPOPOU	8.2.1	TTCAPS	2.10.1
LPOUT	8.2.2	TVCLS	5.5
		TVINP	5.4
MKBUFF	3.9	TVOPIN	5.2
MOVE(10S)	3.3	TVOUT	5.3
		TVST	5.3.1
OPEN(10S)	3.2		
OUT	5.3.3	UPOC	8.2.3
OUTPUT	3.3		
		WLIN	6.4.10
RDBLCK	4.8		
RDBYTE	2.10.4	WRBLCK	4.7
RDIGIT	2.10.2		
RDINT	2.10.3		
RDKEY	7.6		
REGINT	2.5.11		
RESET	5.6		
SPACE	2.3		
STKTST	2.7.2		

Appendix C - Summary of Changes

Except where otherwise stated all changes in function look place prior to the release of the NewBrain in May 1982

Hardware

References to other than Model A NewBrains in the body of this technical manual should be ignored. Some of the hardware ports for the models M and V are implemented in the expansion interface module. Latest details of these are given in Appendix F.

System parameters

The following new system parameters have been added:

<u>Address</u>	<u>Length</u>	<u>Name</u>	<u>Value</u>
6E	3	IOPON	JP Entry to old IOPON routine
71	3	IOPOFF	JP Entry to old IOPOFF routine
74	2	DEFNF	Default numeric format parameter for BASIC to supply to maths pack.
75	1	STR11	Current default graphics stream number for use by PLOT
77	2	CHRRROM	Memory address of start of 2K character pattern look up for text plotting in high resolution.

OS

On power-up the check for a ROM at address A000H will indicate a ROM is present and the power-on routine continues in this ROM. A check is made to see if there is a ROM at address 8000H (contained in an expansion module) and if so the power-on routine continues there (this facility is used to enter the paged memory operating system - POS - in the expansion interface module).

In order for power-up to continue in the 8000H address 0 of this ROM must have bit 7 set to zero, additionally ROM address 2 must not equal 8 (this curious restriction is necessary because in some NewBrains the A000H ROM "wraps round" into the 8000H address space when no ROM is present in this space).

The ROM at A000 contains an intercept to provide new Z - code routines. Details of these and the other Z - code routines are given in Technical Note 10.

This ROM also inserts a new ROM device table and sets its address in DEVTAB. The new device table substitutes new entry points for devices 0 and 4 (TVIO and TLIO) and a new device, device 11 (GRAPH).

The ROM resets the BASIC restarts (RST 32 and RST 40) in order to intercept BASIC and allow for the PLOT command and graphics functions (ARC, MOVE etc).

Power-on concludes by setting STR11 to zero (indicating no default graphics stream yet), DEV0 to 4 (TLIO), CHRRROM to 0B800H (a 2K table is found at this location) and jumping to BASIC via UP.

Appendix C

IOPON and IOPOFF are now called directly at locations 006E and 0071. These routines cater for power switching in battery powered NewBrain models, and are therefore not really relevant to the A/AD models. The routine ACINIT has been removed - its functions are now catered for in the paged memory operating system. NEWIO has been replaced, but does not differ in functional specification or use of own memory. KLOOK has been rewritten to facilitate the production of foreign language keyboard versions. KBII0 and KBWIO use 3 bytes of own memory in later releases of the software: location IX + 1 is not used but is reserved; location IX + 2 is used as an accent parameter. KLOOK will expect IX + 2 to be available for its own use, so IX must be appropriately set up on entry to this routine (the amended versions of LIIO, TVIO, TLIO, KBII0 and KBWIO all do this automatically). Prior to first call the accent parameter must be set to 0. Note that on exit from KLOOK with CY and Z set HL → KBMODE. This latter fact is used by RDKEY!

XIO

In the own memory vector locations IX + 2 and IX + 3, previously unused and used as follows:

- IX + 2 Accent parameter (foreign keyboard versions only)
- IX + 3 ORDEP - original depth = number of display lines held in buffer prior to opening of a graphics stream linked to this character stream.

XIO now takes account of the difference between ORDEP and DEP. LICLS no longer exits via IOPOFF. The video buffer start is now forced to an address of the form $128n + (LL/20)$ (to allow for certain behaviours in the hardware).

The routines REMCUR, WRNC, SHOCUR and WRC have been altered so that the criteria for the appearance or non-appearance of the cursor on the line display are identical to those previously applying to the screen display.

XEDIT has been altered to support character sets 3 and 4 allowing 30 lines on the screen. User control of TVMD1 via the CONTROL W code is restricted by masking to bits 3, 1 and 0.

TVCLS now prevents movement of video areas in memory except while the screen is blanked.

BASIC

PLOT routines are provided which automatically generate the appropriate character strings for the graphics device driver.

DEFNF is set on entry for free format with 8 significant figures. BC is loaded from DEFNF prior to calling maths output when no user formatting ([] - notation) is used.

Appendix D

Rules for writing programs compatible with Expanded NewBrains

On the face of it the only sort of program that can be written for an unexpanded NewBrain is a BASIC program. However, such a program can RESERVE some memory, POKE machine code into it and CALL it. We restrict ourselves here to discussion of four sorts of program: BASIC programs, machine code subroutines to BASIC programs, Device Drivers and User programs.

BASIC programs

basic programs will run identically in expanded systems so long as no use is made of PEEK, POKE and CALL and identical device drivers are connected.

NOTE that the device number in an OPEN statement defaults to the "default back-up store device", which in the unexpanded NewBrain A/AD is device 1 (TAPE 1). In disk systems it is the disk file system device 12. The device number should therefore be specified if a particular device is required.

All communication with hardware must be via the input/output system e.g. PEEKing and POKEing the screen memory or the printer buffer is not compatible with expanded systems. If input/output functions not implemented in the system are required then a standard format NewBrain Device Driver should be written to perform the functions (see below).

Use of Graphics

A maximum 220 screen lines of text graphics should be requested and the amount requested should be a multiple of 10 lines. It should be assumed that if 10n lines are requested then 24-n lines of characters are available in the scrolling character area at the top of the screen (certain configurations provide 25-n lines, but only 24-n are guaranteed to be present).

When opening the editor screen on which the high resolution graphics picture is to be imposed, space for a minimum of 25+5.25 character lines should be requested. More than this will be required if the graphics FILL function is used and how much more can only be determined by experiment. If a program is to allow arbitrary shapes to be FILLED then an ON ERROR trap should be incorporated to intercept failures to fill complex shapes.

No assumptions should be made as to whether or not a dot plotted on the boundary of the graphics area or up to one physical screen dot's distance from it will or will not actually appear on the screen.

Do not assume PEN(x) is implemented for $x \geq 7$.

No major assumptions should be made about the "smoothness" of diagonal lines or arcs.

There are special rules for adapting a BASIC program for implementation in ROM. A software package for this purpose is to be released. Any BASIC program can be adapted but if it is desired to implement part of the program in object code then that part should not contain any of the following commands, which cause object code to be deleted: STOP, RUN, NEW, LOAD, MERGE, RESERVE, CLEAR, DELETE, and END; furthermore the program must never run out of memory.

Appendix D

Machine code subroutines

An area of memory may be RESERVED and a machine code subroutine POKEd into it and CALLED. Such a program may reference the BASIC internal datastructure, any data structures of its own in the RESERVED area and may interface with the operating system by Z-code calls only. This is explained under "CALL statement" and "OS routines" in the NewBrain Handbook.

A machine code subroutine must not reference OS datastructures, as these differ in expanded systems.

No assumptions should be made about the internal format of floating point numbers - Maths Pack Z-code routines should be used to manipulate these.

It is best to produce position-free or relocatable code for machine code routines, as the base of RESERVED areas may vary.

For compatibility the following Z-code routines should not be used in programs intended to run on unexpanded NewBrains: FPHLF, FPIDZ, BICML, RDBYTE, RDINT, RDNSP, BLKIN, BLKOUT and FSTRM (i.e. Z-codes ≥ 59). Calls to BLKIN and BLKOUT should in any case not be made with $BC \geq 256$ or $HL \geq 40960$ or $HL < 8192$.

Device Drivers

A device driver may be constructed, POKEd into RESERVED memory and activated by adding it to the device table (see Technical Notes 7 and 9).

For compatibility, such a driver should not request an own memory area greater than 8184 bytes. (In expanded systems, additional working space can be obtained from the memory management system; and in any case more of the RESERVED area may be used if needs be).

Since a device driver is likely to interface with the hardware no definite guarantee of compatibility can be given - in particular two different device drivers may act in contradiction in their demands on the hardware. However, all hardware functions present in unexpanded NewBrains remain present in expanded ones, so modification, if necessary, is not likely to be difficult.

The method of activating a device driver (given in Technical Note 9) does differ in expanded systems, though the necessary changes are not great. Device drivers will generally have to be relocated to run on expanded systems.

User programs

User programs as defined in the body of this manual can be implemented in expanded systems. The entry parameters to the program itself and to the MKSPC and RETSPC routines are different but re-implementation is, as with the device drivers, not difficult.

Short-cut

A utility is provided for expanded systems which cause them to reconfigure themselves as Unexpanded. If this utility is run then any program written for an unexpanded system can be run completely unmodified. It cannot, however, take advantage of the expansions. Such a program must not make any assumptions about the memory space 8000H-9FFFH.

MEMBRAN PROCESSOR MODULES A/AD

TECHNICAL SUMMARY

Interfaces, Signal Description and Connector Pinout.

CIRCUIT
PIN
TYPE
SIGNAL

SIGNAL DESCRIPTION

Power

- Ground for supply flex screen.
- 1 0
- 2 I +12I Source for 12V internal rail. Accepts 13.0 to 14.5V applied. Consumes about 110mA.
- 3 I -12I Source for -12V and -5V internal rails. Accepts -14.0 to -12V applied. Consumes about 25mA.
- 4 Absent (Accepts polarising blank in power lead plug).
- Ground return for all internal rails.
- 5 I GND
- 6 I +5I Source for 5V internal rail. Accepts 6.5V to 7.5V applied. Consumes about 900mA.

Tape 1 and 2

- 1 I 1FTI Input from tape recorder earphone jack. Loaded by 10nF in series with 39K. Input voltage >500mV required
- 2 O OTTI Output to tape recorder microphone jack. 25mV nominal. Slew rate 4msec/V. 250 ohms source impedance.
- 3 TMSI Relay contact for tape motor. Open circuit DC voltage must be < 20V.
- 4 TMSI Relay contact for tape motor. Open circuit DC voltage must be < 20V.
- 5 TDPLFO Absent (Accepts polarising blank in tape lead plug).
- 6 GRND Ground return for tape signals via flex screen to microphone jack ground only.

Communications

- Received serial data. Input > 2.5V (binary zero) or < 2.5V (binary one).
- 1 I SRDDK
- Nominally not clear to send (CTS or RFS). Used as handshake by software. Characteristics as for pin 1.
- 2 I SCTSD
- Absent (Accepts polarising blank in communications lead plug).
- 3
- Transmitted serial data. Output > 8V (binary zero) or < -8V (binary one) for 3K ohms load. Power up leaves binary zero (contra V24 spec) until stream opened to device.
- 4 O SDO
- Nominally not request to send (RTS). Used as handshake by software. Characteristics as for pin 4 (No power up caveat).
- 5 O SRTSD

Pin	SIGNAL TYPE	CIRCUIT Mnemonic	SIGNAL DESCRIPTION
9	T	D6	D6 from 280A.
10	T	D7	D7 from 280A.
11	T	A11	A11 from 280A.
12	T	A10	A10 from 280A.
13	T	A8	A8 from 280A.
14	T	A9	A9 from 280A.
15	T	A12	A12 from 280A.
16	T	A7	A7 from 280A.
17	T	A3	A3 from 280A.
18	T	A2	A2 from 280A.
19	T	A1	A1 from 280A.
20	T	A0	A0 from 280A.
21	T	D0	D0 from 280A.
22	T	D1	D1 from 280A.
23			Unused. Early machines have a signal connected to this pin. Connect 10K ohm pull-up to 5V. *(HP11).
24	T	D2	D2 from 280A.
25	T	A5	A5 from 280A.
26	T	A6	A6 from 280A.
27	I	RAMENB	Not RAM enable. Logic zero routes 280 memory request to RAM, conditional on pin 48 signal. Unconnected state biased to logic one. *(HP3).
28	I	EXRM2	External A15 signal. Used by paging circuits in the expansion interface module *(HP2).
29	I	EXRM1	External A14 signal. Used as under pin 28 *(HP1).
30	I	EXRM0	External A13 signal. Used as under pin 28 *(HP0).

PIN	SIGNAL	TYPE	CIRCUIT FUNCTION	SIGNAL DESCRIPTION
31	I	ROMOV		Not address override. Logic zero causes A13, A14, A15, to be replaced by external A13,A14,A15.*(GRQ).
32	OC	BUSRQ		Not Bus request as supplied to 280A (BUSRQ).
33	O	M1		Not M1 from 280A (M1)
34	O	RST		Not reset as supplied to 280A (RST).
35	O	REFSH		Not refresh from 280A (REFSH)
36	I	WAIT		Not wait as supplied to 280A (WAIT).
37	T	A4		A4 from 280A.
38	O	BUSAK		Not bus acknowledge from 280A (BUSAK).
39	T	A15		A15 from 280A.
40	T	WR		Not write from 280A (WR)
41	OC	INT		Not interrupt as supplied to 280A (INT).
42	T	RD		Not read from 280A (RD)
43	I	NMI		Not non-maskable interrupt as supplied to 280A (NMI) *(HISLT).
44	O	HALT		Not halt from 280A (HALT) *(MPSL).
45	T	MREQ		Not memory request from 280A (MREQ).
46	T	IORQ		Not input/output request from 280A (IORQ).
47	I	PRTOV		Logic one inhibits internal I/O port decode. Unconnected state biased to logic zero *(HP6).
48	I	RAMINH		Logic one inhibits 280A request to internal RAM. Unconnected state biased to logic zero. *(HP5)
49	O			± 5volts (up to 200mA).
50	O	BUSRQ		Bus request as generated by video circuit *(HP4)
Abbreviations:				
	D0-D7	Data bits 0 to 7	T	Tristate
	A0-A15	Address bits 0 to 15	OC	Open collector
	I	Input	*	Signal function changes on paged expansion highway.
	O	Output		

NEWBRAIN, INPUT/OUTPUT, PORT MAP

NOTE:

1. The NewBrain without expansion interface module decodes port address lines A0 - A4 inclusive. Reference to port 'nx' where x (integral) lies between 0 and 31 inclusive produces the same result for all integral values of n from 1 to 8 inclusive.
2. The addition of the expansion interface module causes decoding of port address lines A0 - A7 inclusive. There is then no 'wrap-around'.
3. Grundy Business Systems will make every effort not to use ports from 32 to 63 inclusive.
4. Some ports are further decoded by the address lines A8 - A15. This is indicated in the relevant port description below.

Abbreviations:

P	Processor modules A or AD
EI	Expansion Interface module
DC	Disc Controller module
NC	Network Controller module
API	Autonomous Peripheral Interface module
ME	Memory Expansion module
R	Read
W	Write
D0 - D7	Data bits 0 to 7
A0 - A15	Address bit 0 to 15

Port	No.	Status		Module	Description
		Read	Write		
0		R/W	EI	Not used.	
1		W	EI	Load enable register 2. R reads and loads all ones.	D0: zero enables user data bus interrupt and also parallel latched data output for centronics printer interrupt. D1: one enables ADC conversion complete interrupt and also calling indicator interrupt. D2: one enables serial receive clock into multiplier input of DAC and signals data terminal not ready. D3: one enables 50K baud serial data rate to be obtained i.e. CTC input clock of 800KHz. Zero selects 107.692KHz. D4: one enables serial receive clock to sound output summer, and also selects serial input from the printer port. Zero selects serial input from the comms port. D5: one enables second bank of four analogue inputs (voltage, non-ratiometric), i.e. CH4-7, and enables sound output, zero selects CH0-3. D6: one enables serial transmit clock to sound output summer, and also selects serial output to the printer port. Zero selects serial output to the comms port. D7: ninth output bit for centronics printer port.
2		W	EI	Load page registers. R reads all ones, but has no other effect. Which of sixteen 8 bit registers is written to is selected by address bits A12, A15, A14, A13, (most to least significant). Contents written are given by A11, D6, D5, D4, D3, D2, D1, D0.	
3		W	EI	Load latched parallel output register (centronics printer port data output). R reads and loads all ones.	

Port No.	Read/Write	Status	Module	Description
4	W		P	Reset clock interrupt. R likewise and reads ones. Data irrelevant.
5	W	EI		Load DAC. R reads and loads all ones. On the first 200 modules the data bits are reversed, i.e. D7 is the least significant bit and D0 the most significant bit.
6	R/W		P	Communicate with COP 420W micro-controller.
7	W		P	Load enable register 1. R reads and loads all ones. D0: zero enables frame frequency clock interrupts. D2: one enables video display. D4: zero asserts 'request to send' on communications port. D5: one asserts binary one data on communications port. D7: one asserts binary one data on printer port. D1, D3 and D6 are not currently used.
8	W		P	Sets ninth bit of video address counter. Data is loaded into first 8 bits of video address counter. Likewise R reads and loads all ones.
9	W		P	Loads first 8 bits of video address counter. R loads and reads all ones.
10	W		P	As 8
11	W		P	As 9
12	W		P	Loads video control register. R loads and reads all ones. D0: one reverses video over entire field, ie. black on white. D1: zero generates 128 characters and 128 reverse field characters from 8 bit character code. One generates 256 characters from 8 bit character code. D2: zero generates 320 or 640 horizontal dots in pixel graphics mode. One generates 256 or 512 horizontal dots in pixel graphics mode. D3: zero selects 256 characters expressed in an 8 x 10 matrix, and 25 lines (max) displayed. One selects 256 characters in an 8 x 8 matrix, and 31 lines (max) displayed. D6: zero selects 40 character line length. One selects 80 character line length D4, D5, D7 are currently unused.
13	W		P	As 12
14	W		P	As 12
15	W		P	As 12
16	R/W	EI		W starts conversion of channel 0 or 4 (same radiometric input ANINO). R reads conversion result of ch0 or 4 (ANINO).
17	R/W		EI	As 16 for ANINI (ch1 or 5).

NEUBRAIN, INPUT/OUTPUT, PORT MAPNOTE:

1. The NewBrain without expansion interface module decodes port address lines A0 - A4 inclusive. Reference to port 'rx' where x (integral) lies between 0 and 31 inclusive produces the same result for all integral values of n from 1 to 8 inclusive.
2. The addition of the expansion interface module causes decoding of port address lines A0 - A7 inclusive. There is then no 'wrap-around'.
3. Grundy Business Systems will make every effort not to use ports from 32 to 63 inclusive.
4. Some ports are further decoded by the address lines A8 - A15. This is indicated in the relevant port description below.

Abbreviations:

P Processor modules A or AD
 EI Expansion interface module
 DC Disc Controller module
 NC Network Controller module
 AEI Autonomous Peripheral interface module
 ME Memory Expansion module
 R Read
 W Write
 D0 - D7 Data bits 0 to 7
 A0 - A15 Address bit 0 to 15

Port No.	Read/Write	Status	Module	Description
0	R/W	EI	EI	Not used.
1	W	EI	EI	Load enable register 2. R reads and loads all ones.
				D0: zero enables user data bus interrupt and also parallel latched data output (for centronics printer) interrupt.
				D1: one enables ADC conversion complete interrupt and also calling indicator interrupt.
				D2: one enables serial receive clock into multiplier input of DAC and signals data terminal not ready.
				D3: one enables 50K baud serial data rate to be obtained ie. CTC input clock of 200KHz. Zero selects 107.692KHz.
				D4: one enables serial receive clock to sound output summer, and also selects serial input from the printer port. Zero selects serial input from the comms port.
				D5: one enables second bank of four analogue inputs (voltage, non-ratiometric), ie. ch4-7, and enables sound output, zero selects ch0-3.
				D6: one enables serial transmit clock to sound output summer, and also selects serial output to the printer port. Zero selects serial output to the comms port.
				D7: ninth output bit for centronics printer port.
2	W	EI	EI	Load page registers. R reads all ones, but has no other effect. Which of sixteen 8 bit registers is written to is selected by address bits A12, A15, A14, A13, (most to least significant). Contents written are given by A11, D6, D5, D4, D3, D2, D1, D0.
3	W	EI	EI	Load latched parallel output register (centronics printer port data output). R reads and loads all ones.

Port No.	Read/Write Status	Module	Description
4	W	P	Reset clock interrupt. R likewise and reads ones. Data irrelevant.
5	W	EI	Load DAC. R reads and loads all ones. On the first 200 modules the data bits are reversed, i.e. D7 is the least significant bit and D0 the most significant bit.
6	R/W	P	Communicate with COP 420M micro-controller.
7	W	P	Load enable register 1. R reads and loads all ones. D0: zero enables frame frequency clock interrupts. D1: one enables video display. D4: zero asserts 'request to send' on communications port. D5: one asserts binary one data on communications port. D7: one asserts binary one data on printer port. D1, D3 and D6 are not currently used.
8	W	P	Sets ninth bit of video address counter. Data is loaded into first 8 bits of video address counter. Likewise R reads and loads all ones.
9	W	P	Loads first 8 bits of video address counter. R loads and reads all ones.
10	W	P	As 8
11	W	P	As 9
12	W	P	Loads video control register. R loads and reads all ones. D0: one reverses video over entire field, i.e. black on white. D1: zero generates 128 characters and 128 reverse field characters from 8 bit character code. One generates 256 characters from 8 bit character code. D2: zero generates 320 or 640 horizontal dots in pixel graphics mode. One generates 256 or 512 horizontal dots in pixel graphics mode. D3: zero selects 256 characters expressed in an 8 x 10 matrix, and 25 lines (max) displayed. One selects 256 characters in an 8 x 8 matrix, and 31 lines (max) displayed. D6: zero selects 40 character line length. One selects 80 character line length D4, D5, D7 are currently unused.
13	W	P	As 12
14	W	P	As 12
15	W	P	As 12
16	R/W	EI	W starts conversion of channel 0 or 4 (same ratio-metric input ANIN01. R reads conversion result of ch0 or 4 (ANIN01).
17	R/W	EI	As 16 for ANIN1 (ch1 or 5).

Port No.	Read/Write	Module	Status	Description
18	R/W	EI	As 16 for ANIN2 if ch0 - 3 selected or ANINV unamplified if ch4 - 7 selected.	
19	R/W	EI	As 16 for ANIN3 if ch0 - 3 selected or ANINV amplified if ch4 - 7 selected.	
20	R	P & EI	Read status register 1.W causes bus contention.	
			D0: fixed at one - indicates excess of 24 or 48. Obsolete (P).	
			D1: one indicates power up from 'cold' - necessary in battery machines with power switching (P).	
			D2: one indicates analogue or calling indicator interrupts.	
			D3: zero indicates centronics printer (latched output data) port interrupt.	
			D4: zero indicates parallel data bus port interrupt.	
			D5: zero indicates frame frequency clock interrupt.	
			D6: zero indicates ACIA interrupt.	
			D7: zero indicates interrupt from micro-controller COP420M.	
21	R	EI	Read status register 2. W causes bus contention. If EI not connected appears identical with port address 20.	
			D2: one selects normal video on power up (white on black), zero selects reversed video (appears as D0 on the first two hundred EI's).	
			D3: one indicates power is being taken from the mains supply.	
			D4: one indicates that 40 column video is selected on power up. Zero selects 80 column video.	
			D6: one indicates that a video display is required on power up.	
22	R	D	Read data input register 1 (user input register 1). W causes bus contention.	
			D0: received serial data from communications port.	
			D1: zero indicates 'clear-to-send' condition at communications port.	
			D5: logic level tape input.	
			D7: zero indicates 'clear-to-send' condition at printer port.	
23	R/W	P & EI	Read or write to user data bus. If EI not connected appears identical with port address 21 and W will cause bus contention.	
24	R/W	EI	Read status and load control registers of the ACIA.	
25	R/W	EI	Read the receive data register and load the transmit data register of the ACIA.	
26		EI	Unused.	
27		EI	Unused.	
28	R/W	EI	Communicate with channel 0 of the CTC.	
29	R/W	EI	Communicate with channel 1 of the CTC.	
30	R/W	EI	Communicate with channel 2 of the CTC.	

Port No.	Read/Write	Module	Description
31	R/W	EI	Communicate with channel 3 of the CTC.
255	W	EI/DC/ NC	Load paging status register. R reads all ones. Modules each contain a paging status register. EI is selected when A8 is one, DC when A9 is one and NC when A10 is one. D0: one enables paging circuits. D1: unused. D2: one sets local A16 to one (ie, causes second set of 8 page registers to select addressed memory). D3: zero selects multi-processing mode. Among other effects this extends the page registers from 8 to 12 bits in length. D4: one isolates the local machine. This is used in multi-processing mode.

Appendix G - Technical Notes

Technical Notes may contain information that has been duplicated elsewhere in the manual but is repeated here for your convenience. Further Technical Notes will be issued from time to time and should be inserted at the back of this manual.

Technical Notes

- 1 VF display character set
- 2 NewBrain keyboard codings
- 3 NewBrain keyboard codings (France)
- 4 NewBrain printer interface
- 5 Memory Allocation
- 6 NewBrain BASIC - General Description
- 7 IOS
- 8 Use of own memory by standard device drivers
- 9 Patching a device driver
- 10 Operating system routines
- 11 Access to screen display characters
- 12 NewBrain BASIC reserved words
- 13 Keyboard sequences
- 14 NewBrain BASIC - Technical Specification
- 15 The NewBrain Graphics Package
- 16 Screen Editor - Technical Specification

NewBrain

Technical Note 1 Issue 1

(NewBrain AD)

VF display character set

There are 128 characters, pictured below.

Of these only 65 can be displayed using NewBrain device drivers 3(VF display editor) and 4(combined screen and VF display editor). These 65 are VF display characters 32-94 (the ASCII characters 32-94), VF display character 95 (the £ sign, screen character 228) and VF display character 104 (this is used for characters "unrecognisable" to the VF display). Drivers 3 and 4 recode the character set so code 168 is displayed as "£", codes 97-122 (ASCII lower case letters) are displayed as VF characters 65-90 (their upper case equivalents), and codes 0-31, 95, 96, 121-167 and 169-254 are displayed as "unrecognisable".

All the characters can, however, be displayed if devices 3 and 4 are not in use, by poking them into the appropriate area of memory. Z80 address 62 corresponds to the right-most position in the VF display, 63 to the next left from it and so on.

VF display position 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

N	E	W	B	R	A	I	N	B	A	S	I	C				
---	---	---	---	---	---	---	---	---	---	---	---	---	--	--	--	--

Memory address 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62

When the memory has been set up the display must be activated by the command:

POKE 59, 160

This command must be repeated whenever the display contents are changed.

Example:

To display the message "NEWBRAIN BASIC ":

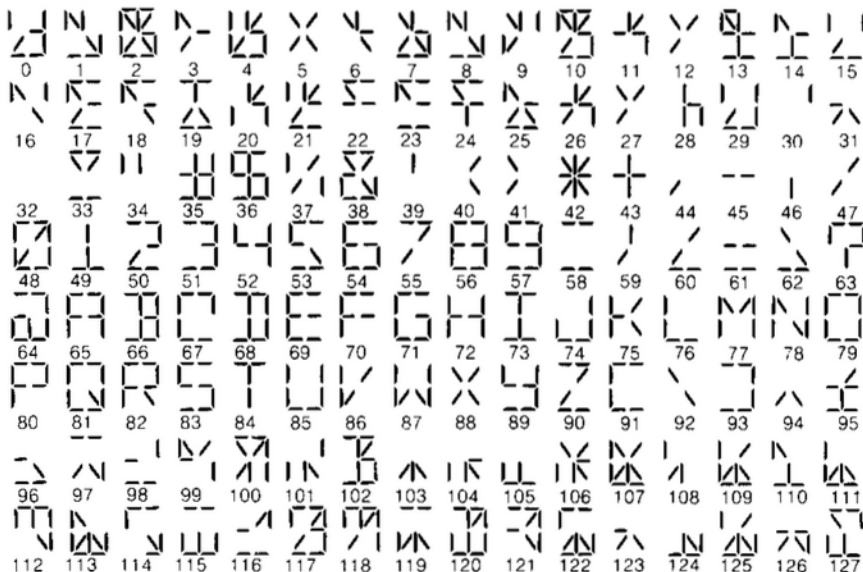
```
10 AS="NEWBRAIN BASIC"
20 FOR I=1 to 16: POKE 78-I, ASC(MID$(AS, I)): NEXT I
30 POKE 59, 160
40 END
```

If the top bit (bit 7) of the code is set then the character will flash.

Example:

To fill the VF display with flashing "X"s:

```
FOR I=62 to 77: POKE I, 128+ASC("X"): NEXT I
POKE 59, 160
```



NewBrain

Technical Note 2 Issue 1

(NewBrain A/AD UK)

NewBrain keyboard codings

The keyboard can be accessed by the user via devices 5 and 6 or by a screen or line editor device (0, 3 or 4). The values returned by the keys are shown in the diagram. The values returned may be modified by "keyboard attributes" (shift-lock, graphics-shift, secondary-graphics and tertiary-graphics) in force at the time of access. Which attributes are in force is determined by the "keyboard mode", a single value in the range 0-9. A separate keyboard mode is maintained by each device 5 or 6, but a single mode value is maintained by the keyboard software itself for all accesses of the keyboard by editor devices. The mode value for editor devices is set by the user entering control 0 - control 9 or control * (these set the mode to the same number, control * to value 0). The mode is set for keyboard devices by PUTting to the device, according to the table:

Mode	PUT value for devices 5 and 6	Control key for editor devices
0	0 or 8	0 or *
1	1 or 9	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	-	8
9	-	9

Modes 8 and 9 are not available to devices 5 and 6.

The modes set the attributes as follows:

Mode	Keyboard attributes in force
0	
1	shift-lock
2	graphics-shift
3	shift-lock, graphics-shift
4	secondary-graphics
5	shift-lock, secondary-graphics
6	graphics-shift, secondary-graphics
7	shift-lock, graphics-shift, secondary-graphics
8	graphics-shift, tertiary-graphics
9	shift-lock, graphics-shift, tertiary-graphics

The keyboard attributes modify returned values only as detailed below:

shift-lock:	The unshifted keys A-Z return the capitals A-Z (i.e. keys normally returning codes 97-122 return 65-90).
graphics-shift:	The shifted keys =, A-Z, (-,_) and + (which normally produce codes 64-94 respectively) produce graphics codes 224-254.
secondary-graphics:	The keys =, A-Z, (-,_) and + when used with graphics (which normally produce codes 128-158, respectively) produce graphics codes 160-190.
tertiary-graphics:	The keys =, A-Z, (-,_) and + when used with graphics produce graphics codes 192-222.

value of key

value of shifted key

value of key with control

value of key with graphics

Control and graphics of 0-9 and * return value 0 to keyboard devices 5 and 6, and do not return values to editor devices.

49 33 1	50 34 2	51 35 3	52 36 4	53 37 5	54 38 6	55 39 7	56 40 8	57 41 9	58 42 0	59 43 ()	60 44 27 155	61 45 =	62 46 0 128	63 47 - _	64 48 28 156	65 49 27 15
13 81 Q	119 87 W	101 69 E	114 82 R	116 84 T	121 89 Y	117 85 U	105 73 I	111 77 O	112 80 P	61 64 =	45 92 - _	27 15 ESCAPE	9 16			
17 145	23 151	5 133	18 146	20 148	25 153	21 149	9 137	15 143	16 144	0 128	28 156					
CONTROL	97 65 A	115 83 S	100 68 D	102 70 F	103 71 G	104 72 H	106 74 J	107 75 K	108 76 L	59 58 :	43 94 +	13 13 NEWLINE	3 16			
	1 129	19 147	4 132	6 134	7 135	8 136	10 138	11 139	12 140	193 96	30 158					
SHIFT	122 90 Z	120 88 X	99 67 C	118 86 V	98 66 B	110 78 N	109 77 M	44 60 127 123	46 62 124 125	47 63 /	SHIFT	0 95 video text	0 0			
	26 154	24 152	3 131	22 150	2 130	14 142	13 141									
GRAPHICS	REPEAT	12 31 HOME	17 1 INSERT	32 32	8 24 .	11 14 !	10 2 1	26 25 -	STOP							
		30 20	22 21	32	28 0	0 18	0 19	29 0								

NewBrain

Technical Note 3 Issue 1

(NewBrain A/AD FRANCE)

NewBrain keyboard codings

The keyboard can be accessed by the user via devices 5 and 6 or by a screen or line editor device (0, 3 or 4). The values returned by the keys are shown in the diagram. The values returned may be modified, in the case of certain keys, by first typing an accent, circumflex " or tréma ". The values returned may also be modified by "keyboard attributes" (shift-lock, graphics-shift, secondary-graphics and tertiary-graphics) in force at the time of access. Which attributes are in force is determined by the "keyboard mode", a single value in the range 0-9. A separate keyboard mode is maintained by each device 5 or 6, but a single mode value is maintained by the keyboard software itself for all accesses of the keyboard by editor devices. The mode value for editor devices is set by the user entering control 0 to control 9 or control * (these set the mode to the same number, control * to value 0). The mode is set for keyboard devices by PUTting to the device, according to the table:

Mode	PUT value for devices 5 and 6	Control key for editor devices
0	0 or 8	0 or *
1	1 or 9	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	-	8
9	-	9

Modes 8 and 9 are not available to devices 5 and 6.

The modes set the attributes as follows:

Mode	Keyboard attributes in force
0	
1	shift-lock
2	graphics-shift
3	shift-lock, graphics-shift
4	secondary-graphics
5	shift-lock, secondary-graphics
6	graphics-shift, secondary-graphics
7	shift-lock, graphics-shift, secondary-graphics
8	graphics-shift, tertiary-graphics
9	shift-lock, graphics-shift, tertiary-graphics

When a circumflex accent or tréma is typed, the value zero is returned for keyboard devices 5 and 6, and no value is returned to editor devices. A coded value is set for the device driver to indicate that the accent has been typed, and if the next key typed is one of those in the tables given below, a modified value is returned.

After circumflex:	A	â, code 205
	E	ê, code 210
	I	î, code 213
	O	ô, code 217
	U	û, code 222
	space	ˆ, code 251
After tréma:	I	ï, code 214
	U	ü, code 223

The keyboard attributes modify returned values only as detailed below:

shift-lock:	The unshifted keys A-Z return the capitals A-Z (i.e. keys normally returning codes 97-122 return 65-90), and the numeric keys 0-9 (which normally produce codes 204, 38, 208, 34, 39, 40, 248, 209, 33, and 225) return the numbers 0-9 (i.e. codes 48-57).
graphics-shift:	The shifted keys, A-Z and -, which normally produce codes 65-90, 95 respectively) produce graphics codes 225-250, 255.
secondary-graphics:	The keys, A-Z and -, when used with graphics (which normally produce codes 129-154, 159, respectively) produce graphics codes 161-186, 191.
tertiary-graphics:	The keys, A-Z and -, when used with graphics produce graphics codes 193-218, 223.

value of key

value of shifted key

value of key with control

value of key with graphics

Control of 0-9, and the ~ key, return value 0 to keyboard devices 5 and 6, and do not return values to editor devices.

38 49 1	208 50 2	34 51 3	39 52 4	40 53 5	248 54 6	209 55 7	33 56 8	225 57 9	204 48 0	41 253)	45 95 .	60 62 <
97 65 A	122 90 Z	101 69 S	114 82 R	116 84 T	121 89 Y	104 72 H	106 74 J	107 75 K	108 76 L	109 77 M	221 37 U	13 13 NEWLINE
CONTROL	113 81 Q	115 83 S	102 70 F	103 71 G	104 72 H	106 74 J	107 75 K	108 76 L	109 77 M	221 37 U	0 40	36 94 \$
SHIFT	119 87 W	120 88 X	99 67 C	118 86 V	98 66 B	110 78 N	44 63 0	59 46 :	58 47 :	61 43 =	SHIFT	27 0 ESCAPE
GRAPHICS	12 31 HOME	17 1 INSERT	32 32 32	32 32 32	32 32 32	8 24 --	11 14 1	10 2 1	26 25 --	STOP	29 0	0 0
REPEAT	30 20	22 21	32 32 32	32 32 32	32 32 32	28 0	0 18	0 19	29 0	29 0	29 0	0 0

New Brain

Technical Note 4 Issue 1

NewBrain printer interface

The NewBrain serial printer interface is a simple RS232C/V24 interface running on 3 wires – DATA, READY and GROUND. Data is transmitted as 8 data bits with 2 stop bits and no parity. The communications protocol is a "ready/busy handshake" in which the printer indicates that it is ready to receive data by an ON condition (also known as a SPACE, "0", binary 0, positive or high voltage condition) of the READY line, and indicates that it is not ready to receive data (i.e. busy) by an OFF condition (also known as a MARK, "1", binary 1, negative or low voltage condition).

Connecting up a printer

An RS232/V24 serial interface printer is usually connected via a 25-way D-type connector. The actual wiring varies from printer to printer and general instructions are given here. The printer manual should be consulted to determine what the correct connections are. Note that a fully specified V24 interface (such as that on the NewBrain) cannot be damaged by a mis-wiring of the connector; but that Grundy Business Systems Ltd cannot be held liable for any damage which might occur to a printer or a NewBrain if they are mis-wired, or for any consequences of it, and therefore a check should be made with the printer supplier if there is any doubt about the interconnection.

The printer may be configured as a "Data Communication Equipment" (DCE – usually the printer is fitted with a male D-type socket and female D-type plug is required) or, as is usual, "Data Terminal Equipment" (DTE – usually the printer is fitted with a female D-type socket and a male D-type plug is required).

(a) Connecting to a printer configured as DTE

The green wire (DATA) is connected to pin 3 of the D-type ("Received Data", RXD). The screen (GROUND) is connected to pin 7 ("signal ground", SG). The blue wire (READY) is variously connected to pin 11 ("Secondary", "supervisory" or "backward channel") or pin 20 ("Data Terminal Ready", DTR). It may be necessary to make certain other connections within the D-type connector to ensure the printer responds to data – in particular pin 4 ("Request to send", RTS) to pin 5 ("Clear to send", CTS) and pin 20 (DTR) to pins 6 ("Data Set Ready", DSR) and pin 8 ("Data carrier detect", DCD). The red and yellow wires are not used.

(b) Connecting to a printer configured as DCE

The green wire is connected to pin 2 of the D-type ("Transmitted Data", TXD), the screen (GROUND) is connected to pin 7 ("signal ground, SG) and the blue wire (READY) is connected either to pin 5 ("clear to send", CTS) or to pin 6 ("data set ready", DSR). As in the case of a printer configured as DTE surplus signals may have to be mopped up by connecting, for example, RTS to CTS and/or DTR to DSR. The red and yellow wires are not used.

Using the printer

```
Enter
OPEN # 8, 8, "2400"
```

to "open" a 2400 baud printer. If possible the printer should be set to 9600 baud as this is the default value so one need enter only

```
OPEN # 8, 8
```

and in any case it is best to run the interface as fast as possible.

Now switch the printer on.

To list programs to the printer then enter

```
LIST # 8
```

or

```
LIST # 8, 100-300
```

to list lines 100 to 300.

To print data enter

```
PRINT # 8, "hello printer"
PRINT# 8, 2+3; a$(14), x1
```

The syntax is the same as PRINT to the screen, but the TAB function is equivalent to a comma (,) separator regardless of the TAB value. The zone width for , is 8 columns.

The printer driver will automatically issue line feed (character 10) after newline (character 13). An automatic newline, line feed, is issued after 80 characters have been printed if a newline is not entered anyway.

If the user wishes to avoid the auto line feed and print zoning then device 9 should be used instead of device 8 and the printer lead should be plugged into the COMMS socket on the back of the NewBrain instead of the PRINTER socket. TAB and comma (,) will result in the issue of TAB control codes (character 9).

For both device 8 and device 9 control codes can be sent to the printer using PUT statements.

```
e.g. PUT # 8, 12      - for form feed
      PUT # 8, 27, 30 - an escape sequence
```

Details of the Communications protocol

On power-up the DATA line is in the binary 0 condition. It remains in this condition until a printer device

(device 8) has been OPENed. Thereafter it remains in the binary 1 condition between data bytes, even if all printer devices are CLOSED.

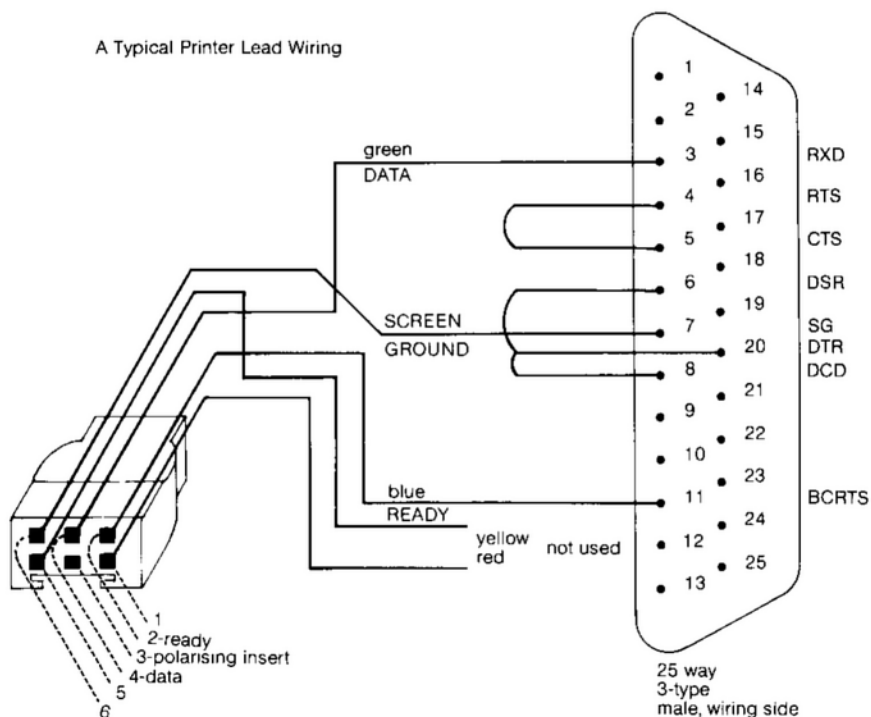
Data is transmitted on the DATA line. Data transmission will not start unless the READY line is in the ON condition. The READY line is sampled before any and every data byte is sent. If no connection is made to the READY line an ON condition is assumed. A transition of the READY line from ON to OFF during transmission of a byte will not inhibit completion of transmission of the byte nor cause an error condition in the NewBrain nor cause retransmission of the byte. The DATA byte is sent as a "start bit" (binary 0), 8 data bits (least significant bit sent first, a binary 0 represents a zero in the corresponding bit of the byte, a binary 1 a one) and two "stop bits" (binary 1). Between bytes the DATA line remains at the stop bit level (binary 1).

Typical voltage levels

A DATA binary 1 is set by the NewBrain at -9 volts. A DATA binary 0 is set at +9 volts.

A transition of the READY line in an ON state to a voltage < -3 volts is interpreted as a transition to an OFF state. A transition of the READY in an OFF state to a voltage > 3 volts is interpreted as a transition to an ON state.

A Typical Printer Lead Wiring



New Brain

Technical Note 5 Issue 1

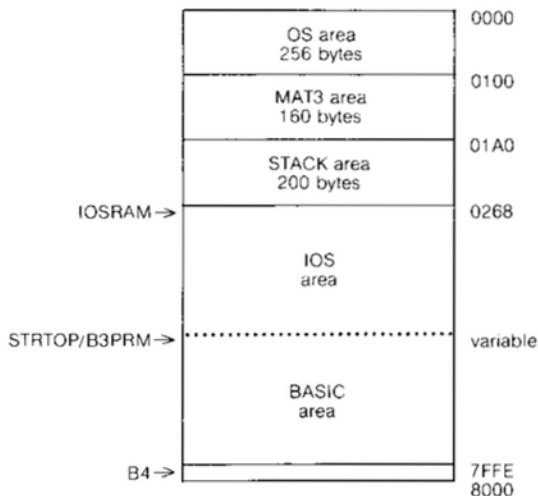
(NewBrain A/AD unexpanded)

Memory Allocation

The software modules which make use of memory are:

- The operating system (OS)
- The Maths Pack (MAT3)
- The Hardware STACK
- The Input/Output System (IOS)
- The BASIC

OS, MAT3 and the STACK are allocated fixed amounts of memory. Memory allocation to IOS and BASIC is dynamic.



Pointers to memory bounds are kept at fixed locations in the OS area.

0062	IOSRAM	The base of the IOS area.
0064	STRTOP	The top of the IOS area.
0004	B3PRM	The base of the BASIC area.
0006	B4	The top of the BASIC area.

Pointers to the base of an area point into the area and pointers to the top of an area point to the first address not used.

B3PRM and B4 are entry parameters to the BASIC. After entry they are not again referenced by the BASIC. IOSRAM and STRTOP are constantly referenced by IOS.

The current software keeps (IOSRAM) static at the top of the STACK area, keeps (STRTOP) equal to (B3PRM) at the dynamic bound between the IOS area and the BASIC area, and keeps (B4) static at two bytes short of the true top of RAM.

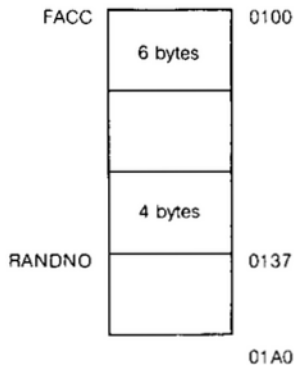
Use of memory areas

1 OS area

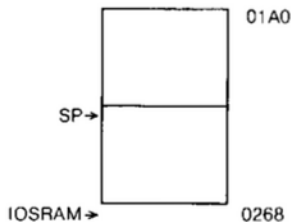
This is used exclusively for static parameters concerned with OS tasks (i.e. interrupt handling, memory management and communication between modules). The locations relevant to this discussion are defined above.

2 MAT3 area

The first six bytes of this area is the floating point accumulator FACC; it is used to pass arguments to and from the maths pack. Four bytes of the area, RANDNO, are used internally as a permanently maintained seed for the random number generator. A pointer to an ASCII string in the maths pack area is returned by the maths routine OUT which converts floating point to text. Otherwise the area is used only internally by the maths pack as a scratch pad.



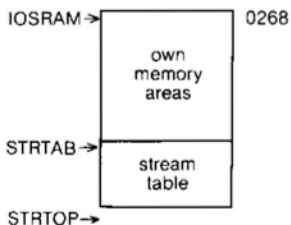
3 STACK area



This is used by the Z80 hardware stack pointer in the usual way. The stack grows downwards in memory, so the active end of the stack (often called the "top of the stack") is below the stack memory area top. In the unexpanded NewBrain the user may move the stack pointer elsewhere but must allow 64 bytes free below it for interrupt handling. Initially the stack pointer is set to the top of the area, i.e. at (IOSRAM).

4 IOS area

IOS maintains a stream table, which is a list of active streams, and for each active stream an "own memory area". Initially there are no streams open and so IOS requires no memory, (IOSRAM) = (B3PRM).



When streams are opened new stream table entries are made at STRTOP and new own memory areas at STRTAB, another OS fixed location parameter. Memory space is taken from BASIC via OS.

0056 STRTAB pointer to base of stream table

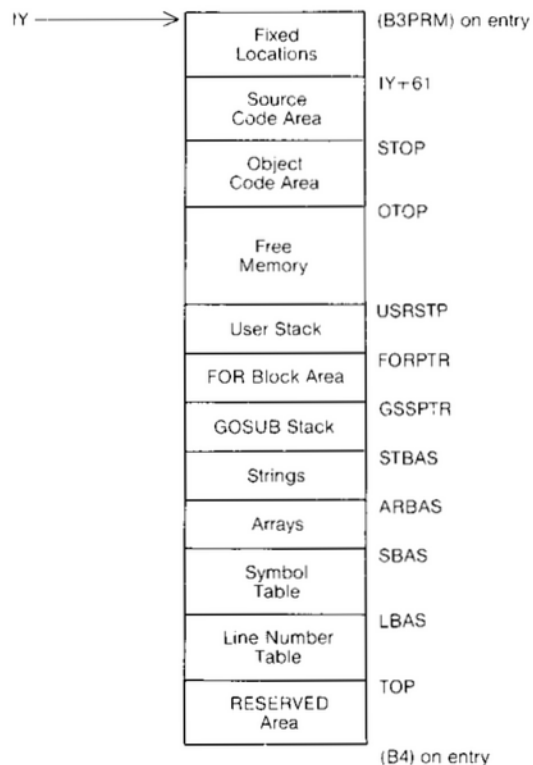
When streams are opened new stream table entries are made at STRTOP and new own memory areas at data structures moved down to close up the area. Memory space is returned to BASIC via OS.

The stream table consists of six byte entries containing pointers to the bases of the corresponding own memory areas:

stream number
device number
port number
reserved
own memory address

5 BASIC area

BASIC uses IY to index the base of its area and the first 61 locations contain general parameters and pointers. The remainder of the area is used to hold the various BASIC data structures:



The various pointers which delineate the areas within the BASIC area are held in the first 61 bytes at offsets from IY:

+0E	STOP	top of source code area
+10	OTOP	top of object code area
+12	USRSTP	user stack pointer
+14	FORPTR	FOR block stack pointer
+16	GSSPTR	GOSUB stack pointer
+18	STBAS	string area base
+1A	ARBAS	array area base
+1C	SBAS	symbol table base
+1E	LBAS	line number table base
+20	TOP	top of line number table

Initially no memory is used by the datastructures so $OTOP = STOP = IY + 61$,
 $USRSTP = FORPTR = GSSPTR = STBAS = ARBAS = SBAS = LBAS = TOP$.

As memory is allocated and deallocated all areas are kept permanently closed up.

When memory is requested by the operating system the lower data structures are moved up into the free area. When it is returned they are moved back down.

When memory is obtained with the RESERVE statement the upper data structures are moved down into the free area.

New Brain

Technical Note 6 Issue 1

(NewBrain A/AD)

NEWBRAIN BASIC - GENERAL DESCRIPTION

NewBrain Basic is an interactive compiler. Each line is compiled separately, and only when execution of a line is required when it has not been previously compiled. This means that in normal circumstances lines are only compiled once, and execution is thus very efficient.

Occasionally, for example when RAM runs out, or when instructions of an exotic nature are used, the object code is deleted. No significant overhead is caused by this, unless there is very little spare RAM or the exotic instructions are used repeatedly.

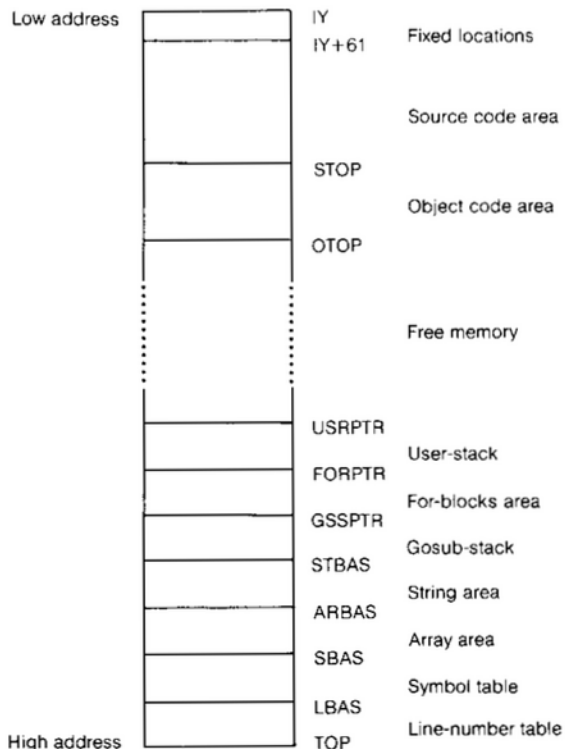
Since the object code consists of pointers into both symbol table and line-number table, these are fixed during execution, and the instructions that alter them (e.g. DELETE, RESERVE, CLEAR, MERGE) cause the object code to be deleted.

The source lines are compressed by exchanging reserved words for 1-byte tokens. The SAVE command outputs this compressed version, reducing time spent saving and re-loading programs. Multi-statement lines are split up in their internal format.

The RAM used by the BASIC has a dynamic lower bound, which the operating system can move by means of the MKSPC and RETSPC routines. The upper bound is also dynamic, but can only be moved by the RESERVE command. Space reserved cannot be given back in a simple manner.

The object code consists of 1-byte operators (Y-codes) which may or may not have arguments, which index a table of run-time routines (X-routines), which perform the run-time operations. Within the BASIC, compilation and execution routines are entirely separate (with the exception of the OPTION and CLEAR statements), except that they can use the upper store management routines, and some basic I/O utilities.

RAM Organisation



Fixed Locations

Displacement from IY	Size		Descriptions
Displacement from IY	Size		Descriptions
6	1	OUTCON	The stream to which output is currently being made.
7	1	INCON	The stream from which input is currently being taken.
8	1	ERRNO	The number of the last error or interrupt that occurred in program execution.
9	2	INPTR	The position from which RDCH etc. are taking their input.
11	1	-	Spare.
12	2	SVINPT	A temporary memory for input positions, used by SVINST and RSINST.
14	2	STOP	Points to Top of Source code.
16	2	OTOP	Points to Top of Object code.
18	2	USRSTP	The base of the User-stack.
20	2	FORPTR	The base of the For-blocks area.
22	2	GSSPTR	The base of the Gosub-stack.
24	2	STBAS	The base of the String area.
26	2	ARBAS	The base of the Array area.
28	2	SBAS	The base of the Symbol table.
30	2	LBAS	The base of the Line-number table.
32	2	TOP	The top of the Basic RAM area, which can be moved down with the RESERVE command.
34	1	FLAGS	Bit 0 is set during running - used by INITOB. Bit 1 is set if interrupts are to be ignored. Bit 2 is set if tokens are not to be expanded during listing (i.e. during SAVE). Bit 3 is set during compilation - used by INITOB. Bit 4 is set whilst compiling a DEF so that VARIAB can watch for formal parameter occurrences. Bit 5 is set if compilation is not to produce code. Bit 6 is set if INPTR is relative to USRSTP currently.
35	2	NCONNO	Count of number of numeric constants encountered, to give each one a unique n/t. Not currently implemented.
37	2	SCONNO	Count of number of string constants encountered, but 14 is always set. Not currently implemented.
39	3	CLINNO	Holds the line-number and displacement where last break of execution in the program occurred.
42	1	BASFLG	Bit 0 = base of arrays (either 0 or 1). Bit 7 = lower base is now fixed if set.
43	1	@TYPE	Temporary variable for EXPN.
44	2	OMOVE	The amount by which the object code has moved during execution of a line.
46	1	PPFLGS	Bit 0 set if in a quoted string. Bit 1 set if after REM or DATA. Bit 2 set during SAVE. Bit 3 set if all KWDS should be tested for, clear if only initial ones should be, or on output whether a token is from KWDS (clear) or OKWDS (set).
47	2	DTINPT	The displacement of the 'DATA pointer' in the source DTLN. Only 1-byte is used, but the other cannot be used for anything else.
49	2	DTLN	The current line number for the 'DATA pointer'. Changes to the LNT entry while executing READ.
51	2	FORVAR	Temporary variable in CFOR, holds the name type of the FOR statement variable whilst searching for the NEXT.
53	2	APAR	Holds the absolute address of the actual parameter (on the user-stack) during execution of DEF statements, accessed by XAATOF.
55	2	FPAR	Holds the name type of formal parameter during compilation of DEF statements, with which VARIAB compares variables it finds.
57	2	ERRLIN	Where control is passed to, on program error. (if this is zero then no trap exists.)
59	2	BRKLIN	Where control is passed to if a break-in occurs. (if this is zero then no trap exists.)

Source Code Area

The source lines of the program and the command line are held here. When old ones are deleted, the others are moved down. New ones are added at STOP. Lines are held in a tokened form, each token referring to a KWD, the syntactic position giving whether it is from KWDS or OKWDS. "Invisible THENS" are used to facilitate this. The ordering of the source lines is not significant. Each one ends in a carriage return, which replaced the ";" in multistatement lines, which are split in the source area. The source code pointers in the line number table point to the first character of the line. Line numbers are not held in the source area but leading and trailing spaces are.

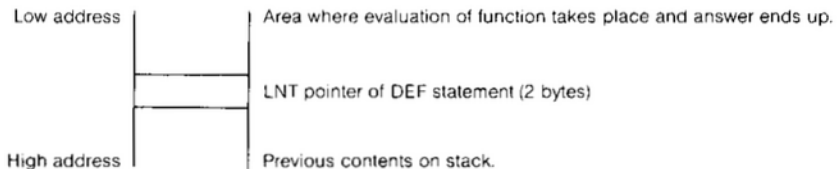
Object Code Area

An object code line is preceded by a 2-byte count giving the length (inclusive) of the object code for that line. There is no significance in the ordering of the lines. New ones are added at OTOP. The object code pointers in the LNT point to the first object code byte (not the count). As the object code has many pointers to the LNT and symbol table, any alterations to these, other than adding symbols at SBAS, require deletion of the object code.

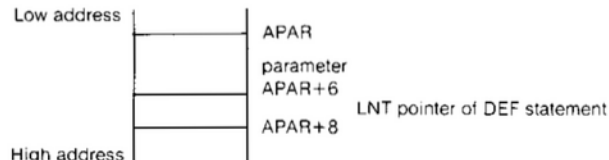
User-stack

The user-stack provides the stack that the object code interpreter uses when a Y-code demands a pushing or a popping. Of note is that for-blocks grow initially in the user-stack and get taken over by FORPTR, and also the following special constructions that may exist on the user-stack on execution of a user-defined function.

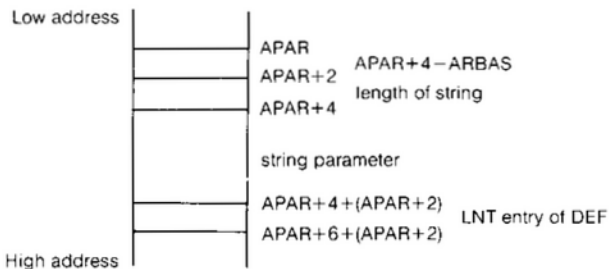
(a) No parameter



(b) Numeric parameter



(c) String parameter



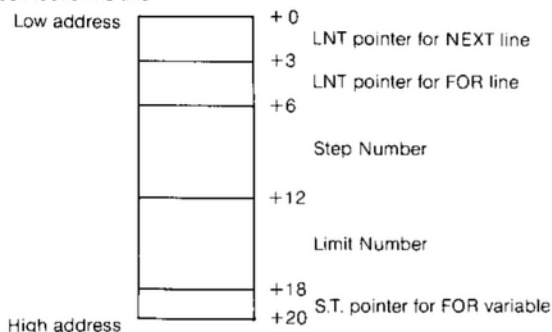
In both (b) and (c) the areas above the top line and beneath the bottom line are the same as in (a). In (c) what is found at APAR must be made to look like a S.T. entry for a string. In all cases the material between the top and bottom lines is removed after evaluation of the user-defined function, so that the answer sits nicely where it should do on the user-stack afterwards.

During Y-code execution strings on the user-stack are represented by a 2-byte count followed by the actual string, the count being the non-inclusive length of the string. Numbers are just represented by their 6-byte floating-point representation.

The user-stack is also used as an input buffer for input from the operating system. Input is collected character by character on the stack until a line has been collected, and then the string of characters including the carriage return, is inverted and referenced by INPTR. This operation is performed by RDLIN.

For-Block Area

A block looks like this



The for-blocks grow out of the user-stack. There are 3-bytes reserved for the LNT entries because under certain circumstances they are converted to line-numbers and displacements. At this time the symbol is converted to a name/type.

GOSUB-Stack

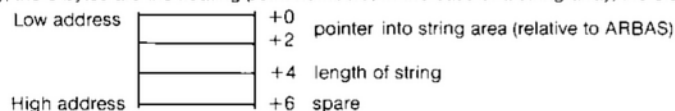
This consists of 3-byte entries, containing the LNT's of the GOSUB statements that have yet to be RETURNed from, the most recent having the lower addresses. 3-bytes are used because at suitable times they are converted to line-numbers and displacements.

String-Area

The string-area contains only the actual strings, which are in no particular order. Pointers into the string-area are relative to ARBAS, as new arrays and symbols can appear at any time. When one is deleted, pointers into the string-area have to be updated (routine MKNULL).

Array Area

The array area contains only the elements of the arrays, each of which is 6 bytes. In the case of a numeric array, the 6-bytes are the floating point numbers; in the case of a string array, the elements are:



The low numbered elements lie at low order addresses, and in the case of a 2-dimensional array, the elements are ordered (0,0), (0,1) ..., (1,0), (1,1)(1,2) so the position of an element is

$BASE + (1st\ SS) * (2nd\ DIM) + (2nd\ SS)$.

Since new symbols can be made at any time, the pointers into the array area are relative to SBAS. When arrays are deleted, the pointers have to be updated (routine ARNULL).

Symbol Table (S.T.)

All symbols are 8-bytes in size. The first two contain the name-type. The low order 10 bits of the name-type "n/t" of a constant are zero. For a variable, the low order 10 bits are a coding of the variable (made by routine GETVNM), which is:

37 * (first letter from 0-25) + (if nothing else then 36, else if it is a digit then 26 + the digit else the second letter from 0-25). The other bits are used as follows:

Bit 10 Spare

Bit 11 = 1 if a user-defined function, otherwise 0.

Bit 12 = 1 for a 1-dimensional array, or if a user-defined function, then it has a numeric parameter.

- Bit 13 = 1 for a 2-dimensional array, or if a user-defined function, then it has a string parameter.
 Bit 14 = 1 if a string constant, string array, string variable, or a user-defined function with a string result, 0 otherwise.
 Bit 15 = 0 if a constant, 1 otherwise.

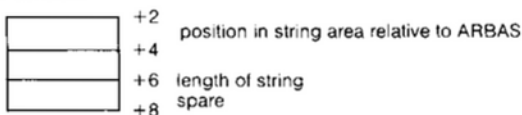
For an array, if both bits 12 and 13 are set, this means that it is not certain yet what number of dimensions the array has. For a user-defined function, if both bits are clear, the function has no parameters.

The other 6 bytes are used as follows:

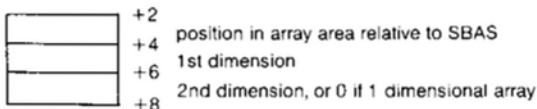
(1) Numeric variable or constant

The F.P. representation of the constant or the current value of the variable is stored.

(2) String variable or constant



(3) Arrays



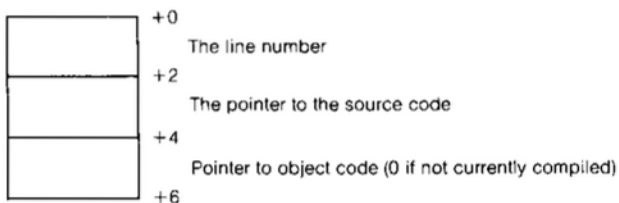
(4) User-defined function

The other 6 bytes are unused.

New symbols are added at the base of the symbol table (SBAS) which reserves the validity of the absolute pointers in the object code. Deletion of symbols (by CLEAR command) does not, so the object code has to be deleted.

Line-Number Table (LNT)

The format of a line-number entry is:



The entries are always held in order by line-number, low numbered lines having low addresses. Multi-statement lines have a separate entry for each statement, each having the same line-number, and are held in the correct textual order. If the line number is 0, the line is, or is part of, the command line.

New Brain

Technical Note 7 Issue 2

(NewBrain A/AD unexpanded)

IOS

The NewBrain hardware and software architecture provides no barrier to I/O bus and memory bus access by a user machine code program.

It is thus possible for the user program to communicate directly with any input/output device. However, this approach is not recommended as NewBrain architecture is complex and protocols for the different devices are various. In any case direct access to a device in a user program is bad software practice – it makes it difficult to transfer the program to different devices and tempts the programmer to take advantage of specific features which may be absent in another environment, even when the same functional device is present.

All NewBrain devices are provided with device driver programs which make device interfaces appear identical. More precisely, all devices appear to be single byte serial. This interface was adopted because it is very simple and straightforward – any physical device can be made to appear as byte serial, while more complex interface protocols, such as block transfer or random access, can still be implemented in the device driver or the user program.

Users are advised when implementing physical devices to write device drivers for them which are in accord with the standard NewBrain interface, or to use one of the standard NewBrain device drivers already implemented.

To further simplify the I/O interface a common set of calling routines, IOS, is provided to enter device drivers. Using these routines all communication with device drivers is via data streams (sometimes known as channels, logical devices or logical units). Routines are provided to initialise a device driver, assign it a data stream, allocate own memory to it, communicate with it and close it down.

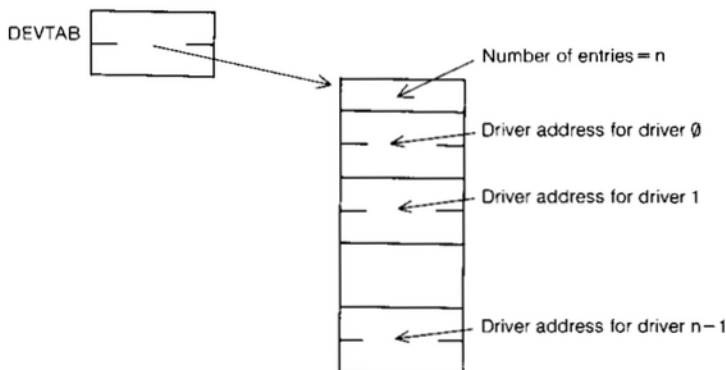
It is recommended that all communication with devices is via these routines – since firstly this avoids the user having to deal with the NewBrain memory management system directly, secondly all communication being via data streams the dependence of a user program on its environment is decreased.

Device Drivers

The existence of a device driver is made known to IOS by the presence of its entry address in the device table. The device table is pointed to by the contents of O/S fixed location DEVTAB.

0058 DEVTAB Address of device table

Fig. 1: Device table



The first byte of the device table gives the number of entries (this may be zero – so at most 255 device drivers can be known to IOS at one time). The subsequent bytes are the entries, each entry being a 2 byte device driver address in standard Z80 low-high format.

In calls to IOS, a device driver is referenced by its "driver number", the number being its entry number in the device table. The standard NewBrain operating system contains a device table in ROM with entries for the NewBrain supplied device drivers. When a user-implemented device driver is required the device table may be kept in RAM.

Fig. 2: NewBrain ROM standard device table

DEFB	12	; Number of entries
DEFW	TV10	; Screen editor
DEFW	TP10	; Tape 1
DEFW	TP20	; Tape 2
DEFW	LI0	; VF display editor

DEFW	TLIO	: Screen editor with VF display
DEFW	KBWIO	: Keyboard
DEFW	KBIO	: Keyboard with immediate return
DEFW	UPIO	: User port
DEFW	LPIO	: Software serial printer
DEFW	JGIO	: Software V24
DEFW	DUMMY	: Null device
DEFW	GRAPH	: High resolution graphics

Each device driver consists of five or more routines; the first five are OPENIN, OPNOUT, INPUT, OUTPUT and CLOSE. The interfaces to OPENIN and OPNOUT are identical – a physical port number and logical parameter string are passed to the routine, which is expected to initialise the device.

When called via IOS the open routine may call MKBUFF in order to create own memory space. Open routines are expected to return to parameter string. The interface to INPUT, OUTPUT and CLOSE is simpler – one byte and a parameter, the port number, are passed to the routine and one byte is returned.

IOS Calling Routines

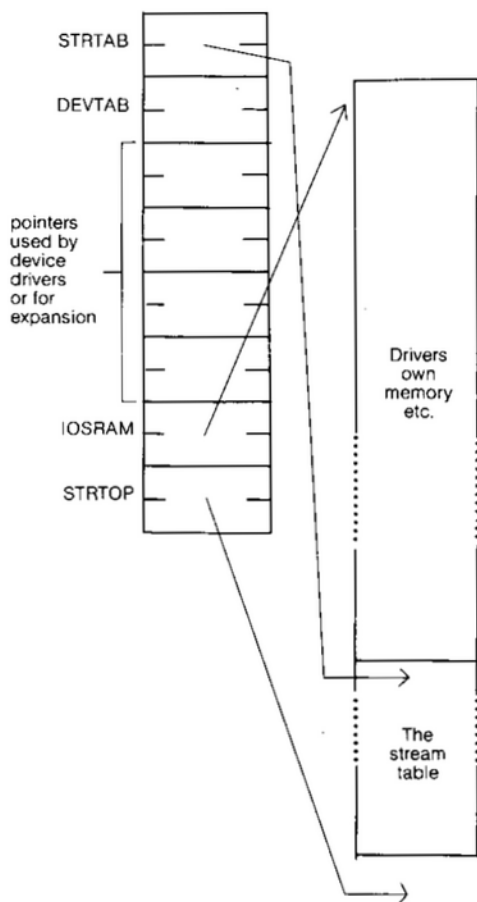
A set of routines is provided to handle all communication between the user program (e.g. BASIC) and NewBrain devices. These routines (OPENIN, OPNOUT, INPUT, OUTPUT, CLOSE etc.) first identify the device driver to be used and then call the corresponding routine from the device driver. OPENIN and OPNOUT allocate a stream to the device, subsequent calls to INPUT, OUTPUT and CLOSE then reference the stream rather than the driver number and port number. As mentioned above device driver open routines may call MKBUFF to allocate own memory to a device, the address of this memory is passed to the device driver on calls to INPUT, OUTPUT and CLOSE. On return from a device driver CLOSE routine the stream and the own memory are deallocated by IOS. A restriction is made on own memory that it should not be absolutely referenced. This is because CLOSEing one device may cause the own memory for another device to be moved. This restriction can be got round in specific cases, but to ensure generality of use programmers are advised to adhere to it. Typically it is assumed own memory is used for parameters, counts, buffers and relative pointers into buffers.

IOS Data Structure

This consists of a set of eight 2 byte pointers at fixed absolute addresses and a memory area which is maintained dynamically. The dynamically maintained area consists of the stream table (at the top of the area) and a general purpose area, the primary use of which is to hold the own memory areas for the device drivers corresponding to open streams. Pointer IOSRAM points to the base of the dynamic area, STRTAB points to the base of the stream table and STRTOP points to the top of the stream table which is the top of the dynamic area. DEVTAB is one of the pointers and the remaining four are for extension or use by device drivers.

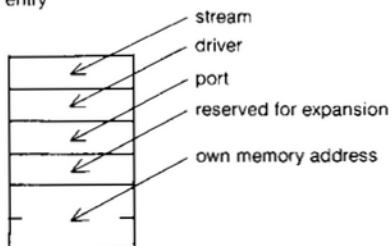
0056	STRTAB	Address of start of stream table
0058	DEVTAB	Address of device table
005A	TVCUR	Used for XIO device driver
005C	TVRAM	Used for XIO device driver
005E	OTHER1	Reserved for extensions
0060	OTHER2	Reserved for extensions
0062	IOSRAM	Base of IOS data structure
0064	STRTOP	Top of stream table

Fig. 3: IOS Datastructure



The stream table consists of six byte entries; the entries contain (from the low address end) the one byte stream number, the one byte driver number, the one byte port number, one byte reserved for expansion and the two byte pointer to the associated own memory area.

Fig. 4: Stream table entry



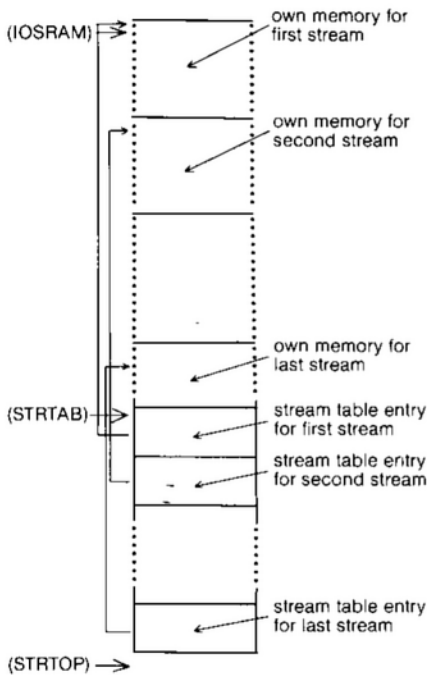
When a stream is allocated by IOS open the stream table entry is added at (STRTOP). When own memory is allocated by MKBUFF it is added at (STRTAB) and the stream table is shifted up. The own memory entry in a stream table entry is set to zero initially and if a driver requires no own memory it should be left at zero. The own memory address may be set by the device driver to point outside the IOS data structure – if this is done it is the user's responsibility, though generally has no ill consequences for IOS. When streams and own memory are removed by CLOSE the IOS data structure is closed up.

When any of these memory adjustments take place all the IOS pointers and the own memory addresses in the stream table entries are adjusted to remain correct.

Notes

(1) The stream table and the own RAM areas are kept in the same order, viz. that in which they were opened (providing MKBUFF is only called from OPEN routines).

Fig. 5: Correspondence between stream table entries and own RAM allocation



- (2) The device table can reasonably be kept in the IOS area.
- (3) The naive programmer doesn't need to know anything about the IOS data structure when writing a device driver or calling an IOS routine, since it is entirely handled by IOS and all parameters of interest to the user are passed on.

IOS Interface

OPENIN and OPNOUT. Calling codes: 032H, 033H, respectively.

Call with

- A = driver number
- D = port number
- E = stream number
- HL = parameter string
- BC = length of parameter string

On return

EITHER

- DE, IX, IY and alternative registers preserved
- CY clear, open succeeded
- BCHL = parameter string returned from device driver
- A destroyed

OR..... CY set, open failed – no stream table entry or own memory made
 BCHL destroyed
 A = error code

BCHL are passed to and from the device driver and not looked at by IOS.

The stream number is checked against the stream table – the open will fail if the stream is already open. The port number is intended primarily as a parameter for the device driver, and is kept by IOS since subsequent calls to IOS INPUT, OUTPUT and CLOSE do not require it as a parameter.

However, the pair consisting of the driver number and port number is checked against the stream table and if already present the open fails.

Possible Error Codes

nodev 106	no such device
stropn 108	stream already open
devopn 107	driver port pair already open
noroom 100	insufficient memory to open
reqerr 109	IOS routine not supported by device

Other error codes may be generated by the device driver.

INPUT, OUTPUT and CLOSE. Calling codes: 031H, 030H, 034H respectively.

Call with E = stream number
 A = data

On return

EITHER..... CY clear, A = data
 OR..... CY set, A = error code

In both cases BCDEHL and other registers preserved.

The data is passed to and from the device driver.

Possible Error Codes

nostrm 105	stream not open
reqerr 109	IOS routine not supported by device

Other errors may be generated by the device driver.

In the case of CLOSE the stream entry and own memory are closed whether or not an error occurred.

MKBUFF. Calling code: 037H

Call with BC = number bytes of own memory required (≠ 0)
 E = stream number

On return

EITHER..... CY clear, HL → own memory
 A destroyed

OR..... CY set, HL destroyed
 A = error code – no room (100)

BCDE and other registers preserved.

MKBUFF may only be called during the execution of the OPEN routine for the stream for which the own memory is being requested and only once during that routine. Because of this restriction in the non-paged-memory operating system one device driver may not open or close another (though it may input from or output to it).

MKBUFF will also crash the system if the stream number is incorrect or if the number of bytes requested is zero.

Device Driver Interface

A device driver must start with a table of one byte displacements to its routines. The first entry in the table is the count, n, of the number of displacements – 1. The remaining (n+1) entries are the displacements. A displacement is set to zero to indicate no such routine is present. IOS uses this table of displacements to call the appropriate routine, the displacements are treated as unsigned eight bit numbers and each is added to the address at which it is found to give the address of the routine.

Example:

DDRV:	EQU \$; a device driver
	DEFB 4	; count of displacements – 1
	DEFB DOPNIN-\$; displacement to OPENIN
	DEFB DOPNOU-\$; displacement to OPNOU
	DEFB DINP-\$; displacement to INPUT
	DEFB DOUT-\$; displacement to OUTPUT
	DEFB DCLS-\$; displacement to CLOSE
	...	

DOPNIN: EQU \$; OPENIN routine

DOPNOU: EQU \$; OPNOU routine

Routine Interfaces:

open in, open out

Called with

D = port number
E = stream number
BCHL = parameter string

Return with

EITHER CY set, A = error code

OR CY clear BCHL = returned parameter string

The routines must preserve IY and alternative registers, but need not preserve the main registers or IX.

While the parameter string is only passed by IOS between the caller and the device driver, many NewBrain user programs attach a meaning to it, and so it should be returned as BC = 0 if not used.

input, output, close

Called with

A = data
D = port number
E = stream number
HL → own memory

Return with

EITHER CY clear, A = data

OR CY set, A = error code

These routines must preserve IY and alternative registers but need not preserve the main registers or IX.

Compatibility with Paged Operating System

The interfaces (between the user program and IOS via OPENIN, INPUT etc., and between IOS and device drivers, i.e. the format of a device driver and the parameters for its routines, open, input, etc.) remain the same under PIOS. However the data structure (apart from device driver own-memory) is implemented differently. Routines are available to add device drivers to the system and to make status enquiries. Own memory is requested via MKBUFF, as before, but requests should be limited to a maximum of 8K. Parameter string lengths should be limited to a maximum 256 bytes.

New Brain

Technical Note 8 Issue 1

NewBrain A/AD (unexpanded)

Use of own memory by standard device drivers, TP1IO, TP2IO, TVIO, LIIO, TLIO, LPIO, JGIO, UPIO and DUMMY

Interface

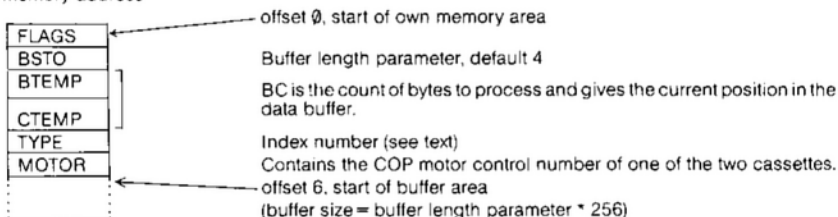
Device driver open routines call MKBUFF to allocate own memory to a device, the address of this memory is passed to the device driver on calls to INPUT, OUTPUT and CLOSE. OPENIN and OPNOUT allocate a stream to the device, subsequent calls to INPUT, OUTPUT and CLOSE then reference the stream rather than the driver number and port number. On return from a CLOSE routine the stream and own memory area are deallocated by IOS. Own memory should not contain absolute addresses as CLOSEing one device may cause the own memory for another device to be moved. Some devices, e.g. video, however, require to make adjustments to the contents of own memory should it be moved. For these a sixth routine MOVE, is implemented with an entry in the device driver entry table.

Cassette Drivers

TP1IO	driver number 1
TP2IO	driver number 2

On open the parameter string is checked and the buffer size is added to the working storage requirement of six bytes to give the total own memory requirement.

Low memory address



High memory address

The FLAGS location uses bits 0 and 7. Bit 0 is set to suppress display of file names on the console stream during OPENIN. Bit 7 is the IN/OUT flag set to 0 by OPENIN and to 1 by OPENOUT.

The index number TYPE is used on OUTPUT to write the block number to tape and on INPUT used to verify the block sequence.

Display Editor Drivers

TVIO	Video display editor	driver number 0
LIIO	Vacuum fluorescent display (VF) editor	driver number 3
TLIO	Video display editor with concurrent VF display	driver number 4

No distinction is made between OPENIN and OPENOUT for these devices and once opened these devices may be written to or read from.

In the discussion which follows the "current line" is the line in which the cursor currently rests (whether or not it is displayed).

As well as keeping the own memory area up to date the editors use the fixed locations TVRAM, TVCUR, TV0, TV1 and TV2 to communicate with the frame interrupt routine.

000D	TV0	Video flash clock, 1 byte
000E	TV2	Editor cursor flags, 1 byte
000F	TV1	Character at cursor, 1 byte
005A	TVCUR	Absolute address of cursor, 2 bytes
005C	TVRAM	Absolute address of current video own memory, 2 bytes

TV2, the editor cursor flags, are used to control video displays:

Bit	
0-3	Reserved
4	1 = Blank video for this frame only
5	0 = Cursor is character 127 1 = Cursor is character 95
6	1 = Cursor currently displayed 0 = Cursor not currently displayed
7	0 = Blank video

Low memory address

Offset to CSTO
TVMD1
Accent
ORDEP
DEP
LL
EL
FLAGS
EXFLGS
LN
FRM
WIN
INPB
INPC
CHAR
DCHR
CSTO
CSTO
BSTO
ESTO
DSTO
FLGSTO
0
0
Space
Space

- 0
- 1 TV mode value written to hardware video control register at each frame interrupt.
- 2 Accent code for foreign language issue NewBrain editors.
- 3 Original depth, no. of display lines held in buffer.
- 4 Depth, no. of display lines held in buffer excluding any high resolution graphics area.
- 5 Line length, no. of display characters per line
- 6 Excess+line length, total bytes per line
- 7 Flags
- 8 Extra flags
- 9 Line number, count of lines from top of display to current line
- 10 Frame, count of lines within buffer to first display line (offset 0)
- 11 Window, no. of characters in current line to start of 16 character VF window
- 12 Initial B value at start of input line
- 13 Initial C value at start of input line
- 14 Temporary storage
- 15 Temporary storage

For video devices with EXCESS 24, sufficiently many unused locations to ensure that video buffer starts on a $128n+(LL/20)$ boundary

Storage for registers

C y coordinate of the cursor within the current line

B x coordinate (offset 0)

E character count of the last line of the current line

D the number of full screen lines in the current line

Flag store

Display buffer 1st location

Display buffer

End bytes

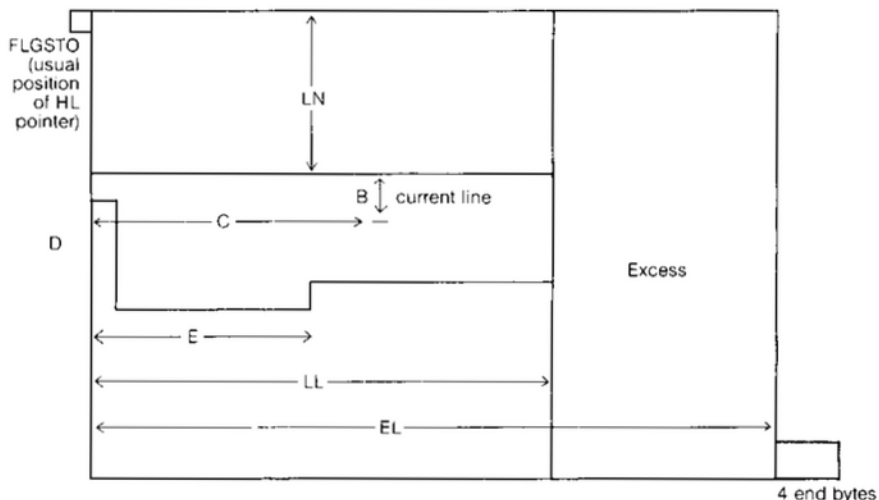
High memory address

The three flag locations are used as follows:

Variable	Flag	Bit	Indicates
FLAGS	SFTFLG	7	1 = Attribute on
	IFLG	6	1 = Insert mode
	CNSFLG	5	This is the console device
	CSIFLG	4	Reserved
	XYFLG	3	User sending cursor xy coordinates
	YFLG	2	0 = expecting x coordinate from user 1 = expecting y coordinate from user
EXFLGS	ESCFLG	1	Escape mode
	TVMFLG	0	Control W mode
	SNDCCFLG	7	User has requested the character at cursor position
	SNDXYFLG	6	User has requested xy coordinates of cursor
	SNDYFLG	5	0 = send x coordinate 1 = send y coordinate
	CURSHOFLG	4	Slow cursor
FLGSTO		3-0	Reserved
	TVFLG	7	Driver supports video display
	CHRFLG	6	Characters for sending in input mode
	LIFLG	5	Driver supports VF display
	INPFLG	4	Editor in input mode

ONFLG	3	One line display set for VF and if depth = 1
PGEFLG	2	Page return mode
HLDFLG	1	Hold mode
CRFLG	0	Carriage return mode

The values of the registers may be shown graphically as follows:



All cursor and memory positional information is held as offset zero. So if the cursor is positioned at the extreme left of the first line of the current line B and C contain zeros. The memory address of the cursor position can be calculated from

$$HL + LN * EL + B * EL + C + 1$$

where HL contains FLGSTO address, LN is the line number of the current line in the display buffer (offset 0) and EL is the total bytes per line, characters and excess.

If B, C and LN are zero the result is HL+1, the first buffer location.

Serial Device Drivers

LPIO	Software serial printer driver	driver number 8
JGIO	Software serial general purpose bi-directional V24	driver number 9

On open the parameter string is checked, BC is set to 4 and MKBUFF (make buffer) is called. The four bytes of own memory are set up as follows:

Receive Baud Rate Parameter	0 offset] JGIO only — LPIO only
Print Line Length set to 80	1	
Print head position initially set to 0	2	
Transmit Baud Rate Parameter	3	

The value of the baud rate parameters is calculated by dividing the baud rate requested into the maximum allowed rate (19200). E.g. 9600 baud gives a baud rate parameter of 2.

User Port

UPIO driver number 7

This driver inputs directly from and outputs directly to the Z80 port specified on open. It does not use own memory.

Dummy Device

DUMMY driver number 10

This device is intended for use in programs where I/O requests are made to a stream but for some reason no I/O operations are required on that stream. It does not use own memory.

New Brain

Technical Note 9 Issue 1

NewBrain A/AD (unexpanded)

Patching a device driver

This note explains how a new device driver may be patched into the NewBrain (unexpanded, i.e. unpagged) operating system. Most of the note consists of an example: patching the software serial driver (JGIO) to receive 7-bit data with even parity instead of 8-bit data with no parity. To understand the note first read Technical Note 7 (IOS). To understand the example in detail reference must be made to the listing of JGIO.

First the device table must be rewritten in RAM. An additional device driver may be added by increasing the entry count by one and adding to the end an additional entry being the address of the new device driver. Alternatively (as in the example below) an old device driver may be replaced by simply reassigning its device table entry to the address of the new driver. If the driver is to be in RAM to run under BASIC then memory space should be reserved for it and the appropriate machine code entered. Reassigning the fixed location DEVTAB to point to the new device table completes the patch.

Example:

Page 2 is the Assembly code for the patched driver.

Page 4 is the BASIC program which patches it in.

ADDR	CODE	STMT	SOURCE STATEMENT
		0001	PSECT ABS
7C36		0002	ORG 31798
E435		0003 JGOP	EQU 0E435H
E49A		0004 JGOUT	EQU 0E49AH
E4E4		0005 JGIN	EQU 0E4E4H
		0006	
7C36	04	0007 NJGIO:	DEFB 4 ; COUNT OF ENTRIES
7C37	05	0008	DEFB NJGOP-\$; OPEN
7C38	04	0009	DEFB NJGOP-\$; OPEN
7C39	0B	0010	DEFB NJGIN-\$; INPUT
7C3A	05	0011	DEFB NJGOUT-\$; OUTPUT
7C3B	07	0012	DEFB NJGCLS-\$; CLOSE
		0013	
7C3C	C335E4	0014 NJGOP:	JP JGOP
7C3F	C39AE4	0015 NJGOUT:	JP JGOUT ; USE OLD ROUTINES
7C42	B7	0016 NJGCLS:	OR A
7C43	C9	0017	RET
		0018	
7C44	CDE4E4	0019 NJGIN:	CALL JGIN ; COLLECT 8 BITS
7C47	B7	0020	OR A ; SET PARITY FLAG ON CONTENTS OF A
7C48	EA4F7C	0021	JP PE NJ1 ; IF EVEN PARITY
7C4B	3E96	0022	LD A, 150 ; ERROR CODE FOR ILLEGAL PARITY
7C4D	37	0023	SCF
7C4E	C9	0024	RET ; ERROR RETURN
		0025	
7C4F	CBBF	0026 NJ1:	RES 7, A ; CONVERT TO 7 BIT DATA
		0027	; DELAY NOW FOR ONE BIT TIME
7C51	4A	0028	LD C, D ; BAUD RATE PARAMETER
7C52	060C	0029	LD B, 12
7C54	10FE	0030 NJ6:	DJNZ NJ6
7C56	0D	0031 NJ7:	DEC C
7C57	C8	0032	RET Z ; DELAY OVER
7C58	060E	0033	LD B, 14
7C5A	10FE	0034 NJ8:	DJNZ NJ8
7C5C	18F8	0035	JR NJ7
		0036	
		0037	END

ERRORS = 0000

The assembly code was produced on a Z-80 cross assembler.

The addresses JGOP, JGOUT and JGIN have been found by examining the JGIO entry table indexed from the device table:

0058	DEVTAB	Address of Device Table
------	--------	-------------------------

For example:

we have $PEEK(5 * 16 + 8) + 256 * PEEK(5 * 16 + 8 + 1) = 41041$

So the device table is found at address 41041. JGIO is device 9 and

$\text{PEEK}(41041 + 9 * 2 + 1) + 256 * (41041 + 9 * 2 + 2) = 58415$

So the JGIO entry table is at address 58415. JGOP, JGIN and JGOUT are the second, third and fourth displacements in the entry table so the required addresses are:

$58415 + 2 + \text{PEEK}(58415 + 2) = 58421$ (0E435H) JGOP

$58415 + 3 + \text{PEEK}(58415 + 3) = 58596$ (0E4E4H) JGIN

$58415 + 4 + \text{PEEK}(58415 + 4) = 58522$ (0E49AH) JGOUT

```
5 REM The following BASIC program patches in the code and activates the new device driver:
6 IF TOP < 32000 THEN 30
10 Reserve 998: REM reserve adequate RAM space for new driver and device table
25 DEF FNhx(i) = PEEK(i) + 256 * PEEK(i+1)
30 dt = FNhx(88) : REM DEVTAB pointer
40 FOR i = 0 TO 24 : POKE TOP+i, PEEK(dt+i): NEXT i: REM copy device table to new location in RAM
50 jf = FNhx(TOP+19): REM device table entry for JGIO = pointer to old JGIO driver
60 jg = TOP+30 : REM location of new JGIO driver
70 POKE TOP+19, jg-INT(jg/256) * 256: POKE TOP+20, INT(jg/256): REM put location of new driver
  into device table
80 RESTORE 100 : ad = jg
90 READ x$: IF x$ <> "" POKE ad, FNd(x$): ad = ad+1: GOTO 90: REM poke in new driver
100 DATA 04, 05, 04, 0B, 05, 07, C3, 35, E4, C3, 9A, E4, B7, C9, CD, E4, E4, B7, EA, 4F, 7C, 3E, 9b, 37, C9,
  CB, BF, 4A, 06, 0C, 10, FE, 0D, C8, 06, 0E, 10, FE, 18, F8, *
105 REM hexadecimal conversion functions
110 DEF FNd(x$) = 16 * FNd1(MID$(x$,1)) + FNd1(MID$(x$,2))
120 DEF FNd1(x$) = ASC(x$) < 50 AND ASC(x$)-48 OR ASC(x$) > 57 AND ASC(x$)-55
130 POKE 88, TOP-INT(TOP/256) * 256: POKE 89, INT(TOP/256): REM activate new device table
140 END
```

A short NewBrain BASIC program was written to read the hexadecimal from the cross assembler system object file output via a V24 interface into the target NewBrain and to create the required DATA statements

New Brain

Technical Note 10 Issue 1

NewBrain A/AD unexpanded

Operating system routines

The following list of calling codes describes the routines provided in the NewBrain operating system. They may be invoked by a machine code program using the instruction RST 32 (op. code E7, decimal 231) followed by the calling code in the next byte – this is in contrast to using CALL followed by a two-byte address. They may also be invoked conditionally on the Carry flag being clear, using the instruction RST 40 (op. code EF, decimal 239) followed by the calling code – in contrast to CALL NC.

(A) Mathematics Routines

The Maths Pack routines are principally distinguished by the number of operands required. Those which have no operands are designated M0, and load the relevant value directly into the Floating Point Accumulator, or FACC, in the Maths Pack working area. Those designated M1 require one parameter, in the FACC, and return a result in the FACC. M2 routines need two parameters; the first is found in the FACC and the second is the six-byte floating point number whose lowest byte has its address in DE when the routine is invoked. The remaining Maths routines have special requirements, and are designated M5; they include routines to load and store the FACC area using numbers in 6-byte floating point format, 16-bit positive integer format, or ASCII string format.

An extra interface is provided to the Maths routines via the special calling code ZMATHS. This is especially appropriate for use by programs such as BASIC, which require a stack of working operands. The MATHS routine is invoked by RST 32 or RST 40 with a Maths code in the A register and the HL register pair pointing to a stack area. The Maths code, which is given after the designation for M0, M1 and M2 routines, is used as a reference to one of those routines. If it is an M0 routine, the HL register pair is decreased by 6 and the result stored where it then points. For an M1 routine, HL points to the operand and the result is stored in the same place. For an M2 routine, HL points to the second operand when the MATHS routine is invoked, and the first operand is at (HL+6); the result is stored in the latter location. In each case, on exit, HL points to the result. If any routine encounters an error, the Carry flag will be set. Where any registers are preserved, they are listed in parentheses after the designation, with "F" representing the FACC. On exit from a routine designated M0, M1 or M2, HL points to the FACC.

Calling code

Hex	Decimal	Name	Designation and Maths code	Result
00	0	ZRND	M0:00	random number
01	1	ZPI	M0:02	constant – pi
02	2	ZSTPOS	M0:04	print head position (from RAM location PHPOS)
03	3	ZFPONE	M0:06	constant – one
04	4	ZFPMONE	M0:08	constant – minus one
05	5	ZFPZERO	M0:0A	constant – zero
06	6	ZNOP	M1:80(F)	no action
07	7	ZNEG	M1:82	change sign if non-zero
08	8	ZNOT	M1:84	logical NOT*
09	9	ZABS	M1:86	absolute value
0A	10	ZATAN	M1:88	arc tangent
0B	11	ZCOS	M1:8A	cosine
0C	12	ZEXP	M1:8C	exponential function (e raised to the power)
0D	13	ZINT	M1:8E	greatest integer less than or equal to
0E	14	ZLOG	M1:90	natural logarithm
0F	15	ZSIGN	M1:90	sign: +1, 0 or –1
10	16	ZSIN	M1:94	sine
11	17	ZSQRT	M1:96	square root
12	18	ZTAN	M1:98	tangent
13	19	ZASIN	M1:9A	arc sine
14	20	ZACOS	M1:9C	arc cosine
15	21	ZPEEK	M1:9E	value of the byte found at the memory address given by the operand
16	22	ZADD	M2:40	sum
17	23	ZSUB	M2:42	first operand – second operand
18	24	ZMULT	M2:44	product
19	25	ZDIV	M2:46	first operand divided by second operand
1A	26	ZRAISE	M2:48	first operand raised to the power of second operand
1B	27	ZAND	M2:4A	logical AND*
1C	28	ZOR	M2:4C	logical OR*
1D	29	ZNUMEQ	M2:4E	TRUE if the operands have the same value; FALSE otherwise

1E	30	ZNUMGT	M2:50	TRUE if first operand greater than the second; FALSE otherwise
1F	31	ZNUMLT	M2:52	TRUE if first operand less than the second; FALSE otherwise
20	32	ZNUMNE	M2:54	TRUE if operands do not have the same value; FALSE otherwise
21	33	ZNUMLE	M2:56	TRUE if first operand is less than or equal to the second; FALSE otherwise
22	34	ZNUMGE	M2:58	TRUE if the first operand is greater than or equal to the second; FALSE otherwise
23	25	ZMATHS	MS	loads the FACC from value addressed by HL, sets DE, and calls appropriate M0, M1 or M2 routine, given by Maths code in A, then stores result. CY set and A = 2 if error
24	36	ZINITRAND	MS (F)	the random number generator is set to a standard value
25	37	ZRANDOM	MS (F)	the random number generator is set to the value of the 4 byte page zero location CLOCK
26	38	ZCOMP	MS (F)	CY is set if first operand (HL) greater than the second (DE), clear otherwise. Zero flag is set if operands are equal
27	39	ZFIX	MS	operand addressed by HL is converted to 16-bit positive binary and returned in DE
28	40	ZFLT	MS	16-bit positive binary value in DE is converted to floating point in the FACC. HL points to the FACC
29	41	ZROUND	MS	contents of the FACC are increased by 0.5 and converted to 16-bit positive binary in DE
2A	42	ZINP	MS	ASCII string referenced by DE is converted to floating point, if it is a number. DE is set to the first unused character, HL to the FACC
2B	43	ZLDF	MS (BCHL)	floating point number addressed by HL is loaded into the FACC
2C	44	ZOUT	MS	value in the FACC is converted to an ASCII string in the Maths pack working area, using BC as the format code.** HL is left pointing to the string, with C = number of characters
2D	45	ZSTF	MS (BCHLF)	contents of the FACC are copied into the location addressed by HL
2E	46	ZRE_boo	VS	examines Carry and Zero flag and returns TRUE or FALSE. Enter with bit 0 of B set if NC, NZ should yield TRUE; bit 1 set if C, NZ should yield TRUE; and bit 2 set if Z should yield TRUE
3E	62	ZFPHLF	MS	constant - half
3F	63	ZFPID2	MS	constant - half of pi
40	64	ZBICML	MS	converts value in the FACC to integer part in DE and most significant two bytes of fractional part in BC, CY set if overflow, Z set if positive, NZ if negative

* The logical operations NOT, AND, OR first convert the operand or operands to 16-bit twos complement numbers in the range -32768 to +32767. If this cannot be done an error is returned. A bitwise logical operation is then performed and the result is floated into the FACC.

** The format coding in MC specifies the format type "f" in bits 6, 7 of B, the number of digits "i" for the integer part in bits 0-5 of B and the total digit count "t" in C, as shown below. If the format is inappropriate, the string returned by ZOUT will contain asterisks on overflow or periods on underflow.

f = 0:	Fixed point format Number is rounded to decimal with t digits, t-1 decimal places. Leading zeros represented as spaces, 1 leading space or minus sign, 1 trailing space. Total field width t+3, except when t = i (integer format), t+2.
f = 1:	Free format Number is rounded to t significant figures and output in fixed or exponential format - goes to exponential above 10 ¹ or below 10 ⁻³ . Underflows are converted to ASCII 0. The field width is variable.
f = 2:	Exponential format Number is output as for fixed point format with two digit exponent; total field width t+7.

For all formats must have $10 \geq i > 0$.

(B) Operating System and Input-Output System Routines

If an error is encountered, the Carry flag is set and the A register contains an error number. Some routines always return with Carry clear; for others (e.g. TTCAPS) the Carry flag does not denote an error

Calling code

Hex	Decimal	Name	Registers	Details
2F	47	ZTTCAPS	BCDEHL	Byte in A, if it is "a" to "z", is converted to capital "A" to "Z". Entry: A = data byte E = stream number
30	48	ZOUTPUT	BCDEHL	Entry: E = stream number Exit: A = data byte if CY clear Open stream for input.
31	49	ZINPUT	BCDEHL	Entry: A = driver number D = port number, E = stream number, BC = parameter string length HL = start of parameter string.
32	50	ZOPENIN	DE	Exit: BC, HL = parameter string returned Open stream for output, as OPENIN Close stream
33	51	ZOPENOUT	DE	Entry: E = stream number
34	52	ZCLOSE	DEHL	Checks if remaining stack space is large enough Entry: BC = number of bytes Exit: CY set if insufficient room left on stack to add BC bytes
35	53	ZSTKTST	BCDE	Checks and clears the STOP key flag Exit: if STOP key pressed, A = 0 and CY set Creates an own memory area in IOS space of BC bytes Entry: BC = number of bytes required E = stream number Exit: if CY clear, HL points to start of own memory area provided Note: MKBUFF may only be called by a device driver during OPEN, and should only be called once!
36	54	ZBRKTST	BCDEHL	Gets a byte of coded keyboard data from the COP. Z80 may power down during GETKEY Exit: A = data byte containing keyboard matrix coordinate
37	55	ZMKBUFF	BCDE	Gets a byte of coded keyboard data from the COP if one is available Exit: CY clear, A = data byte as GETKEY CY set if no data byte available
3A	58	ZKLOOK	DE	Converts keyboard matrix coordinate to ASCII or graphics character Entry: A = keyboard matrix coordinate Exit: A = character Note: Page zero location KBMODE is used to determine any currently in force keyboard attributes
3B	59	ZFSTRM	ABCDE	Finds a matching stream number Entry: E = stream number Exit: CY clear, HL points to stream table entry at offset 0
3C	60	ZBLKIN	DE	Inputs a block of bytes from a stream into memory Entry: BC = number of bytes HL = first memory location E = stream number Exit: CY clear, BC = 0, HL increased by BC CY set, BC = number of bytes not input, HL increased by the number of bytes input

Calling code		Name	Registers	Details
Hex	Decimal			
3D	61	ZBLKOUT	DE	<p>Outputs a block of bytes from a memory area</p> <p>Entry: BC = number of bytes HL = first memory location</p> <p>Exit: CY clear, BC = 0, HL increased by BC CY set, BC = number of bytes not output HL increased by number of bytes output</p>
41	65	ZRDBYTE	BDE	<p>Converts an ASCII string or substring to an 8-bit integer</p> <p>Entry: C = string character count HL = location of first character of string, with subsequent characters at higher addresses</p> <p>Exit: C = count on entry less number of numeric characters read A = binary integer HL = address following address of last numeric character read CY set if overflow or non-numeric</p>
42	66	ZRDINT	B	<p>Converts an ASCII string or substring to a 16-bit integer</p> <p>Entry: as for RDBYTE</p> <p>Exit: C, HL as for RDBYTE DE = binary integer</p>
43	67	ZRDNSP	BDE	<p>Scans a string for a non-space character</p> <p>Entry: as for RDBYTE</p> <p>Exit: Z flag set if count C exhausted HL = address of first non-space character found C = count of characters remaining A = first non-space character found, converted to capital "A" to "Z" if it was "a" to "z".</p>

New Brain

Technical Note 11 Issue 1

Access to screen display characters

The NewBrain screen editor provides facilities to enable a user to find characters sited anywhere on the display screen and to place characters directly on to the screen in selected positions. This is done using the control codes detailed below, which may be output to any TV stream using PUT in a BASIC command line or program.

Position cursor at x,y	PUT 22,x,y
Place a character c at position x,y	PUT 22,x,y,c
Find out what character is at position x,y	PUT 22,x,y,20: GET c
move cursor to HOME (1,1)	PUT 12
Move cursor up one (if possible)	PUT 11
Move cursor down one (if possible)	PUT 10
Move cursor left one (if possible)	PUT 8
Move cursor right one (if possible)	PUT 26
Find cursor position	PUT 21: GET x,y

The entire line on which the cursor is positioned may be obtained using

```
PUT 5: LINPUT (" ") a$
```

The entire screen area may be obtained, for instance to copy it to another stream, by a sequence of statements:

```
PUT 3: FOR I=1 to 24: LINPUT (" ") a$:....: NEXT I
```

Enough LINPUT commands must be executed to allow for every line of the screen.

Example: To check if the character to the right of position (p,q) is a blank, and if so move the character at position (p,q) one to the right:

```
PUT 22, p+1,q: GET c  
IF c=32 THEN PUT 8, 20: GET c: PUT 32, c: p=p+1
```

The cursor is positioned one to the right of (p,q) and the character at that position obtained (c). If it is a blank (code 32), the cursor is moved one to the left, and the character code which was there is obtained (c again). A blank is written, which moves the cursor one to the right, followed by the character c.

While instructions of this sort are being processed, the cursor will not normally be shown. If it is desirable to show the cursor, e.g. while developing programs, control code 6 may be PUT to the screen. The cursor will then be displayed at all times until control code 7 is PUT.

The control codes 3, 5, 20 and 21 cause the screen driver to expect one or more calls of GET, INPUT, or LINPUT. It will go on expecting such calls until either the right number of characters have been returned, or any byte is PUT to the screen. E.g. to copy the first 5 lines of the screen to a printer:

```
OPEN # 2, 8: PUT 3  
FOR I=1 to 5: LINPUT # 0, a$: PRINT # 2, a$: NEXT I  
PUT 0: REM cancels effect of PUT 3
```

When LINPUT is used in this way, the default prompt "?-" must be suppressed by using the form LINPUT # s, a\$ or LINPUT (" ") a\$. Otherwise, the prompt which is output to the screen cancels the "send" mode and causes the program to enter the usual keyboard input mode.

New Brain

Technical Note 12 Issue 2

(NewBrain A/AD)

NewBrain BASIC reserved words

NewBrain BASIC reserved words are entokened. The same token may be used for two different words, meaning being obtained from word order.

e.g. PRINT ASC

has character 131 then character 158, PRINT POKE being illegal.

Entokened words in the range 128 to 158 can be input by using the GRAPHICS key. For those in the range 160 to 187 it is necessary for "CONTROL 4" (5, 6 or 7) to be entered first, then use the GRAPHICS key with the character in the table. This gives you the secondary-graphics mode, hence the "2" shown in the table.

The odd one out is character 159 (CALL or LEN). This first requires attributes on (type CONTROL N), then type ESCAPE followed by SHIFT HOME. Don't forget to turn the attributes **off** (SHIFT/ESCAPE).

















NewBrain Reserved Words Table

Token value			"GRAPHICS" and	Token value			"GRAPHICS" and
128	LET	NOT	=	159	CALL	LEN	ATT*
129	LIST	+	A				SHIFT
130	NEW	-	B				HOME
131	PRINT	*	C	160	CONTINUE	VAL	2 = **
132	RUN	/	D	161	DEF	NUM	2A
133	GOTO	--	E	162	CLEAR	ASN	2B
134	GO TO	&	F	163	MERGE	ACS	2C
135	GOSUB	AND	G	164	VERIFY	PEEK	2D
136	GO SUB	OR	H	165	GET	ERRNO	2E
137	IF	<>	I	166	RESUME	INSTR	2F
138	INPUT	<=	J	167	DELETE	ERRLIN	2G
139	FOR	>=	K	168	PUT	TOP	2H
140	NEXT	=	L	169	LINPUT	FREE	2I
141	RETURN	>	M	170	REPORT	CHR\$	2J
142	REM	<	N	171	CONT	STR\$	2K
143	END	RND	O	172	RESERVE	RIGHT\$	2L
144	STOP	PI	P	173		FILES	2M
145	DIM	POS	Q	174		LEFT\$	2N
146	ON	ABS	R	175		MIDS	2O
147	OPTION	ATN	S	176		BASE	2P
148	SAVE	COS	T	177		IN#	2Q
149	LOAD	EXP	U	178		OUT#	2R
150	RANDOMIZE	INT	V	179		#	2S
151	OPEN	LOG	W	180		THEN	2T
152	CLOSE	SGN	X	181	Invisible	THEN	2U
153	?	SIN	Y	182		TO	2V
154	RET	SQR	Z	183		STEP	2W
155	RESTORE	TAN	(184		FN	2X
156	READ	TRUE	-	185		TAB	2Y
157	DATA	FALSE)	186		ERROR	2Z
158	POKE	ASC	+	187		BREAK	2(

















































































* ATT is CONTROL N (attributes on)

** "2" means secondary graphics CONTROL 4 (5, 6 or 7)

NewBrain BASIC Reserved Words Table

BASIC	Token value	Keyboard sequence "GRAPHICS" and	Character set 1	Character set 2
ABS	146	R		
ACS	163	2C		
AND	135	G		
ASC	158	+		
ASN	162	2B		
ATN	147	S		
BASE	176	2P		
BREAK	187	2(	

BASIC	Token value	Keyboard sequence "GRAPHICS" and	Character set 1	Character set 2
CALL	159	ATT SHIFT HOME		
CHR\$	170	2J		
CLEAR	162	2B		
CLOSE	152	X		
CONT	171	2K		
CONTINUE	160	2=		
COS	148	T		
DATA	157)		
DEF	161	2A		
DELETE	167	2G		
DIM	145	Q		
END	143	O		
ERRLIN	167	2G		
ERRNO	165	2E		
ERROR	186	2Z		
EXP	149	U		
FALSE	157)		
FILES\$	173	2M		
FN	184	2X		
FOR	139	K		
FREE	169	2I		
GET	165	2E		
GO SUB	136	H		
GO TO	134	F		
GOSUB	135	G		
GOTO	133	E		
IF	137	I		
IN#	177	2Q		
INPUT	138	J		
INSTR	166	2F		
INT	150	V		
LEFT\$	174	2N		
LEN	159	ATT SHIFT HOME		
LET	128	=		
LINPUT	169	2I		
LIST	129	A		
LOAD	149	U		
LOG	151	W		
MERGE	163	2C		
MID\$	175	2O		

BASIC	Token value	Keyboard sequence "GRAPHICS" and	Character set 1	Character set 2
NEW	130	B		
NEXT	140	L		
NOT	128	=		
NUM	161	2A		
ON	146	R		
OPEN	151	W		
OPTION	147	S		
OR	136	H		
OUT#	178	2R		
PEEK	164	2D		
PI	144	P		
POKE	158	+		
POS	145	Q		
PRINT	131	C		
PUT	168	2H		
RANDOMIZE	150	V		
READ	156	-		
REM	142	N		
REPORT	170	2J		
RESERVE	172	2L		
RESTORE	155	(	
RESUME	166	2F		
RET	154	Z		
RETURN	141	M		
RIGHT\$	172	2L		
RND	143	O		
RUN	132	D		
SAVE	148	T		
SGN	152	X		
SIN	153	Y		
SQR	154	Z		
STEP	183	2W		
STOP	144	P		
STR\$	171	2K		
TAB	185	2Y		
TAN	155	(	
THEN	180	2T		
THEN (invisible)	181	2U		
TO	182	2V		
TOP	168	2H		

BASIC	Token value	Keyboard sequence "GRAPHICS" and	Character set 1	Character set 2
TRUE	156	-		
VAL	160	2=		
VERIFY	164	2D		
#	179	2S		
&	134	F		
*	131	C		
+	129	A		
-	130	B		
/	132	D		
<	142	N		
=	140	L		
>	141	M		
?	153	Y		
-	133	E		
<>	137	I		
<=	138	J		
>=	139	K		

New Brain

Technical Note 13 Issue 1

(NewBrain A/AD)

Keyboard sequences

All 256 characters of the NewBrain character set are obtainable from the keyboard. Mode group 1 is graphics-shift (corresponding to CONTROL 2, 3, 6, 7, 8 or 9, i.e. press CONTROL key and "2" simultaneously). Mode group 2 is secondary-graphics (CONTROL 4, 5, 6 or 7) and mode group 3 is tertiary-graphics (CONTROL 8 or 9).

In the table the following keys are denoted

cntl for CONTROL
sh for SHIFT
grph for GRAPHICS
ATT for CONTROL N (attributes on)
cursor LEFT for ←
cursor UP for ↑
cursor DOWN for ↓
cursor RIGHT for →



































































Some characters can be obtained in more than one way, these sequences are separated by a comma. The character shown in brackets corresponds to character set 2.

Mode group 1 (Shift keys 224–254 control 2, 3, 6...9)

Mode group 2 (Graph keys 161–254 control 4...7)

Mode group 3 (Graph keys 192–222 control 8, 9)

0	cntl/=	()	29	cntl/CURSOR RIGHT	(→)
1	cntl/A,sh/INSERT	(␣)	30	cntl/+,cntl/HOME	(␣)
2	cntl/B,sh/CURSOR DOWN	(↓)	31	sh/HOME	(␣)
3	cntl/C,cntl/NEW LINE	(↵)	32	SPACE BAR	()
4	cntl/D	(␣)	33	sh/1	(!)
5	cntl/E	(␣)	34	sh/2	(@)
6	cntl/F	(␣)	35	sh/3	(#)
7	cntl/G	(␣)	36	sh/4	(\$)
8	cntl/H,CURSOR LEFT	(←)	37	sh/5	(%)
9	cntl/I,cntl/ESCAPE	(␣)	38	sh/6	(&)
10	cntl/J,CURSOR DOWN	(↓)	39	sh/7	(')
11	cntl/K,CURSOR UP	(↑)	40	sh/8,((())
12	cntl/L,HOME	(␣)	41	sh/9)	()
13	cntl/M,sh/NEW LINE, NEW LINE	(↵)	42	*	(*)
14	cntl/N,sh/CURSOR UP	(↑)	43	+	(+)
15	cntl/O,sh/ESCAPE	(␣)	44	,	(,)
16	cntl/P,grph/ESCAPE,grph/NEWLINE	(␣)	45	-	(-)
17	cntl/Q,INSERT	(␣)	46	.	(.)
18	cntl/R,grph/CURSOR UP	(↑)	47	/	(/)
19	cntl/S,grph/CURSOR DOWN	(↓)	48	0	(0)
20	cntl/T,grph/HOME	(␣)	49	1	(1)
21	cntl/U,grph/INSERT	(␣)	50	2	(2)
22	cntl/V,cntl/INSERT	(␣)	51	3	(3)
23	cntl/W	(␣)	52	4	(4)
24	cntl/X,sh/CURSOR LEFT	(←)	53	5	(5)
25	cntl/Y,sh/CURSOR RIGHT	(→)	54	6	(6)
26	cntl/Z,CURSOR RIGHT	(→)	55	7	(7)
27	cntl/[,ESCAPE	(␣)	56	8	(8)
28	cntl/-,cntl/CURSOR LEFT	(←)	57	9	(9)

58	sh/;	(:)	102	F	(f)	146	grph/R	()
59	;	(;)	103	G	(g)	147	grph/S	()
60	sh/,	(<)	104	H	(h)	148	grph/T	()
61	=	(=)	105	I	(i)	149	grph/U	()
62	sh/.	(>)	106	J	(j)	150	grph/V	()
63	sh//	(?)	107	K	(k)	151	grph/W	()
64	sh/=	()	108	L	(l)	152	grph/X	()
65	sh/A	(A)	109	M	(m)	153	grph/Y	()
66	sh/B	(B)	110	N	(n)	154	grph/Z	()
67	sh/C	(C)	111	O	(o)	155	grph/(()
68	sh/D	(D)	112	P	(p)	156	grph/-	()
69	sh/E	(E)	113	Q	(q)	157	grph/)	()
70	sh/F	(F)	114	R	(r)	158	grph/+	()
71	sh/G	(G)	115	S	(s)	159	ATT/sh/HOME	()
72	sh/H	(H)	116	T	(t)	160	2/grph/=	()
73	sh/I	(I)	117	U	(u)	161	2/grph/A	()
74	sh/J	(J)	118	V	(v)	162	2/grph/B	()
75	sh/K	(K)	119	W	(w)	163	2/grph/C	()
76	sh/L	(L)	120	X	(x)	164	2/grph/D	()
77	sh/M	(M)	121	Y	(y)	165	2/grph/E	()
78	sh/N	(N)	122	Z	(z)	166	2/grph/F	()
79	sh/O	(O)	123	grph/.	({)	167	2/grph/G	()
80	sh/P	(P)	124	cntl/.	({)	168	2/grph/H	()
81	sh/Q	(Q)	125	grph/.	(!)	169	2/grph/I	()
82	sh/R	(R)	126	grph//	(})	170	2/grph/J	()
83	sh/S	(S)	127	cntl/.	(~)	171	2/grph/K	()
84	sh/T	(T)	128	grph/=	()	172	2/grph/L	()
85	sh/U	(U)	129	grph/A	()	173	2/grph/M	()
86	sh/V	(V)	130	grph/B	()	174	2/grph/N	()
87	sh/W	(W)	131	grph/C	()	175	2/grph/O	()
88	sh/X	(X)	132	grph/D	()	176	2/grph/P	()
89	sh/Y	(Y)	133	grph/E	()	177	2/grph/Q	()
90	sh/Z	(Z)	134	grph/F	()	178	2/grph/R	()
91	sh/(([)	135	grph/G	()	179	2/grph/S	()
92	sh/-	(\)	136	grph/H	()	180	2/grph/T	()
93	sh/)	(])	137	grph/I	()	181	2/grph/U	()
94	sh/+	()	138	grph/J	()	182	2/grph/V	()
95	sh/VIDEO TEXT	()	139	grph/K	()	183	2/grph/W	()
96	grph/;	()	140	grph/L	()	184	2/grph/X	()
97	A	(a)	141	grph/M	()	185	2/grph/Y	()
98	B	(b)	142	grph/N	()	186	2/grph/Z	()
99	C	(c)	143	grph/O	()	187	2/grph/(()
100	D	(d)	144	grph/P	()	188	2/grph/-	()
101	E	(e)	145	grph/Q	()	189	2/grph/)	()

190	2/grph/+	(ʌ)	234	1/sh/J	(ʝ)
191	ATT/sh//	(Ω)	235	1/sh/K	(ʞ)
192	3/grph/=,sh/0	(Σ)	236	1/sh/L	(♥)
193	3/grph/A,cntl/;	(Δ)	237	1/sh/M	(♠)
194	3/grph/B,cntl//	(Π)	238	1/sh/N	(ñ)
195	3/grph/C,grph//	(÷)	239	1/sh/O	(Ø)
196	3/grph/D	(↓)	240	1/sh/P	(ʘ)
197	3/grph/E	(÷)	241	1/sh/Q	(ʑ)
198	3/grph/F	(ʒ)	242	1/sh/R	(æ)
199	3/grph/G	(ʒ)	243	1/sh/S	(ʒ)
200	3/grph/H	(ʒ)	244	1/sh/T	(ʒ)
201	3/grph/I	(i)	245	1/sh/U	(ʒ)
202	3/grph/J	(j)	246	1/sh/V	(÷)
203	3/grph/K	(á)	247	1/sh/W	(ʒ)
204	3/grph/L	(à)	248	1/sh/X	(ʒ)
205	3/grph/M	(à)	249	1/sh/Y	(ʒ)
206	3/grph/N	(à)	250	1/sh/Z	(œ)
207	3/grph/O	(à)	251	1/sh/{	(^)
208	3/grph/P	(é)	252	1/sh/-	()
209	3/grph/Q	(è)	253	1/sh/)	(°)
210	3/grph/R	(é)	254	1/sh/+	(■)
211	3/grph/S	(i)	255	ATT/cntl/,	(■)
212	3/grph/T	(i)			
213	3/grph/U	(i)			
214	3/grph/V	(i)			
215	3/grph/W	(ó)			
216	3/grph/X	(ò)			
217	3/grph/Y	(ò)			
218	3/grph/Z	(ò)			
219	3/grph/{	(ò)			
220	3/grph/-	(ú)			
221	3/grph/)	(ú)			
222	3/grph/+	(ú)			
223	ATT/sh/VIDEO TEXT	(ü)			
224	1/sh/=	(ñ)			
225	1/sh/A	(ç)			
226	1/sh/B	(ñ)			
227	1/sh/C	(ç)			
228	1/sh/D,sh/*	(é)			
229	1/sh/E	(á)			
230	1/sh/F	(é)			
231	1/sh/G	(i)			
232	1/sh/H	(ó)			
233	1/sh/I	(ú)			

New Brain

Technical Note 14 Issue 1

NewBrain BASIC – Technical specification

1 DATATYPES

1.1 Numbers

These are handled by the mathematics package.

1.1.1 Storage Six bytes are allocated by BASIC for the storage of a number value. The mathematics package maintains and operates on numbers to a precision of 10 or more significant figures in a range of $0... \pm 10^{+150}$.

1.1.2 Output By default output is rounded to 8 significant figures and is in free format (integer, floating point or scientific "E" notation according to value). Output to specific field sizes and formats can be forced by format specifiers for the PRINT statement and the STR\$ function. Output range is $0... \pm 10^{+99}$.

1.1.3 Input Any output format is accepted as input, additionally any number of digits is allowed in the mantissa and spaces may be disregarded.

1.1.4 Constants Any number valid for input may be used as a numerical constant.

1.1.5 Variables Any simple name consisting of a letter or a letter followed by a letter or a digit may be used to denote numeric variables. A variable may hold any valid number.

1.2 Strings

In NewBrain BASIC a string is any sequence of bytes (i.e. numbers in the range 0..255). Bytes often stand for characters, in particular those in the range 32..127 stand for the ASCII printing characters.

1.2.1 Storage Strings can be of any length between 0 and 32767 bytes. An additional overhead of four bytes is required for each string stored. Storage allocation is dynamic (i.e. the length of a string can change during program or command execution).

1.2.2 Output Any string can be output. Input and output devices interpret bytes in different ways. For instance the keyboard screen editor device, which is usually the console for BASIC, interprets 0..31 as control codes, 32..127 as ASCII character codes and 128..255 as mosaic graphics character codes.

1.2.3 Input Any valid string constant may be supplied in response to a BASIC INPUT or READ statement. If the constant does not contain quotation marks ("), commas or TAB (code 09) characters the enclosing quotation marks may be omitted. Comma and TAB characters are used by INPUT to separate consecutive strings. In response to LINPUT any string not containing NEWLINE (code 13) character may be supplied.

1.2.4 Constants Any string not containing a NEWLINE character may be enclosed in quotation marks ("") and used as a string constant in a BASIC statement. Quotation marks within a string constant are denoted by doubled quotation marks ("").

1.2.5 Variables Any simple name followed by a dollar (\$) character may be used as a string variable name. A string variable may hold any string.

1.3 Logical

There is no special logical datatype, but numbers may be used to store logical values. In those cases where a logical value is required for a binary choice, -1 is taken as TRUE and all other values are taken as FALSE. Logical operations are performed bitwise on 16-bit words. In this sense -1 is TRUE, 0 is FALSE and other values from the range -32768..32767 take intermediate truth value; some operations require arguments from this subrange.

1.4 Arrays

Arrays may be of numbers or strings, of 1 or 2 dimensions to a maximum of 5575 elements. Array storage must be reserved by DIMensioning, but each element of a string array may vary in length during program or command execution. In addition to the storage required for the values of the elements there is a storage overhead of 6 bytes for an array and a further overhead of 2 bytes per element for a string array.

There is no provision for input/output of whole arrays.

Individual array elements may be treated the same way as numeric and string scalars for input/output.

There are no array constants.

Any number (or string) variable name may be used for a numeric (or string) array. A variable name thus used may also be used for a scalar of the same type, but may not be used for another array of the other dimension.

2 EXPRESSIONS

2.1 Atomic Expressions

The allowed atomic expressions are:

constant variable name array element function call

2.2 Molecular Expressions

Atomic expressions may be built up to form molecular expressions.

If X and Y are valid atomic expressions then

Unary operation X X binary operation Y (X)

are valid molecular expressions subject to the type restrictions detailed below.

Constant This may be of any string or number constant.

Variable Name This may be any string or number variable name.

Array Element This is

array name (expression) or array name (expression, expression).

The expressions involved must evaluate to numbers within the dimensions of the array. Numbers are rounded to the nearest whole number for this purpose.

Function Call The general form is

function name (arguments)

The number and type of arguments depends on the function, given this, an argument can be any expression of the correct type. Arguments are separated by commas.

2.2.1 Functions with No Arguments

PI	The mathematical constant
RND	A pseudo-random number from the uniform distribution of the unit interval (0,1)
POS	The current position of the printhead on the console output device
TRUE	Evaluates always to -1
FALSE	Evaluates always to 0
ERRNO	The error number of the most recent error or break-in unless cleared
ERRLIN	The line number of line being executed at the time of the most recent error or break-in
FREE	The number of bytes of free store available to but not used by BASIC
TOP	The lowest store address not available to BASIC
FILES\$	The parameter string returned by the most recently OPENed input device

2.2.2 Functions with One Numeric Argument

INT	Integer part (INT(X) is the greatest integer not greater than (X))
ABS	Absolute value (modulus)
SGN	Sign (-1 for negative argument, 1 for positive, 0 for zero)
SQR	Square root
SIN COS TAN	Trigonometric functions
ASN ACS ATN	Inverse trigonometric functions
LOG	Natural logarithm
EXP	Exponential function
PEEK	Contents of memory location whose address is the argument
CHR\$	The string consisting of the single character whose internal code is the argument
STR\$	The string consisting of the numeric output format of the argument
	The format may be forced. The expression STR\$(X[<i>formatter</i>]) produces a string in the format determined by the formatter. Allowed formatters are:
	n Integer format with n digits
	n.m. Fixed point format with n digits before and m after the point
	n.mE Scientific format with n digits before the point, m after and 2-digit exponent a multiple of n
	nF Free format in a field width of n
	In all formats leading zeroes in the mantissa are replaced by spaces, there is a leading space or minus sign and a trailing space.

2.2.3 Numeric Valued Functions with One String Argument

LEN	Length of the string
VAL	The numeric value of the string, if the string happens to be in a valid number input format
NUM	= -1 (TRUE) if the string happens to be in a valid number input format = 0 (FALSE) otherwise
ASC	The NewBrain internal code for the first character of the string (for characters in the ASCII set this coincides with the ASCII code)

2.2.4 Substring Functions

LEFT\$	LEFT\$(X\$,N) extracts the leftmost N character substring
RIGHT\$	RIGHT\$(X\$,N) extracts the rightmost N character substring
MID\$	MID\$(X\$,P) extracts the right hand substring starting at the P'th character of X\$ MID\$(X\$,P,N) extracts the substring of length N starting at character P
INSTR	INSTR(X\$,Y\$) finds the numerical position of the string Y\$ in X\$ INSTR(X\$,Y\$,P) finds the position in MID\$(X\$,P)

2.2.5 User Defined Functions These are numeric or string valued with one or no numeric or string argument. A user defined function name is a simple name preceded by FN. User defined functions must be

declared equal to an expression in a DEF statement.

2.2.6 Unary Operations These all take numeric arguments and yield numeric results.

+ - unary plus and minus
NOT The bitwise Boolean operation

2.2.7 Binary Operations with Numeric Arguments which Yield Numeric Results

+ * / Plus minus times divide
! Raise to a power
< <= - Relational operators
> >= <> (less than, less than or equal, equal, greater than, greater than or equal, not equal)
AND OR The bitwise Boolean operations

2.2.8 Binary Operations with String Arguments which Yield Numeric Results

< < = Relational operators returning Boolean results as in the numeric case.
> >= <> Characters are ordered according to the NewBrain internal codes (which ordering agrees with ASCII and hence with alphabetic order) and strings are ordered by first difference.

2.2.9 Binary Operations with String Arguments which Yield String Results

+ (also &) Concatenation

2.3 Expression Evaluation

Within an expression atomic expressions and expressions in parentheses are evaluated first. Operations are evaluated in the order

+ (unary operations)

* / (binary operations)

& (binary operations)

< <= > >= <>

NOT

AND

OR OR having the least binding power

Mathematical operations and functions are evaluated by the mathematics package. The mathematics package has been specially designed to minimise cumulative rounding errors and maintain stability in recursive and iterative calculation, and to obtain maximal computation speed consistent with ten significant figure accuracy and long term stability.

3 INPUT/OUTPUT

This is performed by the NewBrain operating system.

3.1 I/O Devices

I/O devices are signified by whole numbers in the range 0 to 255. In practice in a program, devices are usually named as numeric variables whose values signify the device. All input/output peripherals are configured by their device drivers as byte serial devices.

3.2 Data Streams

Data streams are signified by whole numbers in the range 0 to 255, though in practice numeric variable names are used. In BASIC, after opening a device on a datastream, all communication with that device is via the datastream. When appropriate devices and device drivers are connected, appropriate control datastreams can be used to implement all current I/O modes including random access.

4 BASIC STATEMENTS, BLOCKS AND PROGRAMS

A BASIC statement generally consists of a keyword followed by certain arguments. A BASIC line consists of statements (simple statements or FOR-block statements) separated by colons. A line may be used as a command, or introduced by a line number as part of a BASIC program. Line numbers are whole numbers in the range 1..65535.

4.1 Simple Statements

4.1.1 Declaration

OPTION BASE 0

Sets the array base globally to 0

OPTION BASE 1

Sets the base globally to 1.

The default base is 0.

DIM array name (dimension)

The dimension must be an appropriate numeric expression or a pair of expressions separated by a comma. Multiple dimensionings separated by commas are allowed.

Scalars are not explicitly declared, arrays need not be. In the latter case a default dimension of (10) or (10, 10) is assumed.

DEF *function name (argument) = expression*

declares a user defined function.

4.1.2 Assignment

LET *assignee = expression*

An assignee may be a variable name or an array element. The assignee must be of the type of the evaluated expression. The keyword LET may be omitted.

4.1.3 Control

IF *expression* THEN *line number*

IF *expression* THEN *line*

In the latter case the keyword THEN may usually be omitted.

GOSUB *line number*

RETURN (also RET)

ON *expression* GOTO *line number list*

ON *expression* GOSUB *line number list*

ON ERROR GOTO *line number* Set error trap

ON BREAK GOTO *line number* Set interrupt trap

RESUME Resume after trap

RESUME *line number*

GOTO *line number*

CONTINUE (also CONT)

STOP

RUN

NEW

CALL *expression, arguments*

This is to call a machine code or other language routine.

4.1.4 Input and Output

OPEN *direction # stream, device, port, parameter string*

Direction is IN or OUT, it is optional and defaults to IN. Stream is an appropriate numeric expression. Device and port are numeric expressions which may be omitted. Parameter string is a string expression which defaults to null. The information is passed to the operating system which assigns the stream to the device and opens it.

INPUT *assignee list*

INPUT # *stream, assignee list*

INPUT (*prompt*) *assignee list*

The *prompt* string expression is used to substitute for the default "?" prompt when the stream is the default (console) stream.

PRINT *print list*

PRINT # *stream, print list*

? is a synonym for PRINT. *Print list* is a sequence of *print items* and *print separators*. *Print items* must be separated by at least one *print separator*, and can be

expression

numeric valued expression [formatter]

TAB (*numeric valued expression*)

The *formatter* controls the numeric output format as in STR\$.

TAB moves the printhead position on the console

Separators are commas and semicolons, these also control the position of the printhead.

LINPUT *assignee*

LINPUT # *stream, assignee*

LINPUT (*prompt*) *assignee*

assign all input until the next NEWLINE (code 13) character from the input stream to the assignee string.

GET *assignee list*

GET # *stream, assignee list*

assign input bytes sequentially to the assignees, string assignees being set to single character strings, numeric assignees to numbers in the range 0..255.

PUT *expression list*

PUT *stream, expression list*

output bytes. If an expression evaluates to a string the byte corresponding to its first character is used, if the expression evaluates to a number then this must be in the range 0..255.

CLOSE # *stream* closes the stream.

LIST *range*

LIST # *stream, range*

These output the current program as text.

Range can be absent or

line number—*line number*

—*line number*

line number

line number—

SAVE # *stream* outputs the program in "entokened" form.

2.2.6 Unary Operations These all take numeric arguments and yield numeric results.

NOT	The bitwise Boolean operation
-----	-------------------------------

+ - * / Plus minus times divide

< <= > >=

Relational operators

AND OR The bitwise Boolean operations

< <= > >=

Relational operators returning Boolean results as in the numeric case

2.2.9 Binary Operations with String Arguments which Yield String Results

2.3 Expression Evaluation

+ - (unary operations)

1

 $\langle \cdot \rangle = \langle \cdot \rangle = \langle \cdot \rangle = \langle \cdot \rangle = \langle \cdot \rangle$

AND

Mathematical operations and functions are evaluated by the mathematics package. The mathematics package has been specially designed to minimise cumulative rounding errors and maintain stability in recursive and iterative calculation, and to obtain maximal computation speed consistent with ten significant figure accuracy and long term stability.

This is performed by the NewBrain operating system.

I/O devices are signified by whole numbers in the range 0 to 255. In practice in a program, devices are usually named as numeric variables whose values signify the device. All input/output peripherals are configured by their device drivers as byte serial devices.

Data streams are signified by whole numbers in the range 0 to 255, though in practice numeric variable names are used. In BASIC, after opening a device on a datastream, all communication with that device is via the datastream. When appropriate devices and device drivers are connected, appropriate control datastreams can be used to implement all current I/O modes including random access.

A BASIC statement generally consists of a keyword followed by certain arguments. A BASIC line consists of statements (simple statements or FOR-block statements) separated by colons. A line may be used as a command, or introduced by a line number as part of a BASIC program. Line numbers are whole numbers in the range 1...65535.

4.1.1 Declaration

Sets the *array base* globally to 0

Sets the base globally to !

DIM *array name* (*dimension*)

Scalars are not explicitly declared, arrays need not be. In the latter case a default dimension of 10, or 10¹⁰ is assumed.

DEF *function name* (*argument*) = *expression*

declares a user defined function.

4.1.2 Assignment

LET *assignee* = *expression*

An assignee may be a variable name or an array element. The assignee must be of the type of the evaluated expression. The keyword LET may be omitted.

4.1.3 Control

IF *expression* THEN *line number*

IF *expression* THEN *line*

In the latter case the keyword THEN may usually be omitted.

GOSUB *line number*

RETURN (also RET)

ON *expression* GOTO *line number list*

ON *expression* GOSUB *line number list*

ON ERROR GOTO *line number* Set error trap

ON BREAK GOTO *line number* Set interrupt trap

RESUME

RESUME *line number*

GOTO *line number*

CONTINUE (also CONT)

STOP

RUN

NEW

CALL *expression, arguments*

This is to call a machine code or other language routine.

4.1.4 Input and Output

OPEN *direction # stream, device, port, parameter string*

Direction is IN or OUT, it is optional and defaults to IN. Stream is an appropriate numeric expression. Device and port are numeric expressions which may be omitted. Parameter string is a string expression which defaults to null. The information is passed to the operating system which assigns the stream to the device and opens it.

INPUT *assignee list*

INPUT # *stream, assignee list*

INPUT (*prompt*) *assignee list*

The *prompt* string expression is used to substitute for the default "?" prompt when the stream is the default (console) stream.

PRINT *print list*

PRINT # *stream, print list*

? is a synonym for PRINT. *Print list* is a sequence of *print items* and *print separators*. *Print items* must be separated by at least one *print separator*, and can be

expression

numeric valued expression [*formatter*]

TAB (*numeric valued expression*)

The *formatter* controls the numeric output format as in STR\$.

TAB moves the printhead position on the console

Separators are commas and semicolons, these also control the position of the printhead.

LINPUT *assignee*

LINPUT # *stream, assignee*

LINPUT (*prompt*) *assignee*

assign all input until the next NEWLINE (code 13) character from the input stream to the assignee string.

GET *assignee list*

GET # *stream, assignee list*

assign input bytes sequentially to the assignees, string assignees being set to single character strings, numeric assignees to numbers in the range 0..255.

PUT *expression list*

PUT *stream, expression list*

output bytes. If an expression evaluates to a string the byte corresponding to its first character is used, if the expression evaluates to a number then this must be in the range 0..255.

CLOSE # *stream* closes the stream.

LIST *range*

LIST # *stream, range*

These output the current program as text.

Range can be absent or

line number – *line number*

– *line number*

line number

line number –

SAVE # *stream* outputs the program in "entokened" form.

LOAD # *stream* MERGE # *stream* VERIFY # *stream*
 LOAD inputs a new current program in textual or in "entokened" form. MERGE merges the input program with the current one. VERIFY checks the current program against the one on the input stream.

LOAD *filename* MERGE *filename*
 SAVE *filename* VERIFY *filename*

LOAD *filename* is equivalent to

OPEN IN # 128, *default backup store device, filename*
 LOAD # 128
 CLOSE # 128

and similarly for others. The filename may be omitted.

READ *assignee list* DATA *constant list*
 RESTORE RESTORE *line number*

These last four statements are for the "internal reader".

4.1.5 Miscellaneous

POKE *expression, expression*

directly addresses machine store.

RANDOMIZE

sets the random number generator to an unknown value.

REM introduces a comment
 REPORT

if an ON ERROR or ON BREAK trap is set outputs the message which would have been output had it not been, and halts program or command execution.

CLEAR CLEAR *list*

release store used by all, or selected, arrays variables and constants.

DELETE *range* release store used by current program
 RESERVE *expression*

reserves free store previously available to BASIC to release for use by user's machine code or other purposes. Statements containing no characters at all are also allowed.

4.2 Blocks

Blocks are sequences of simple statements which may or may not make up program lines, or FOR-blocks which are

for-statement *blocks* *next-statement*

for-statement is

FOR *variable* = *initial value* TO *limit* STEP *increment*

initial value, limit and *increment* are numeric valued expressions.

STEP *increment* may be omitted.

next-statement is

NEXT *variable*

The control variable must be numeric and the same in both the for and in the next statement

4.3 Programs

A program is both a sequence of lines and a sequence of blocks. An END statement should be present.

5 MODE OF OPERATION

A certain input/output device, called the console, is open when BASIC is started up. Input from this device is treated by BASIC either as lines which are commands to be obeyed immediately, or as lines with line numbers which are added to the current program to be obeyed later when the program is run. Reserved words in lines are "entokened" in the program in order to minimise store usage. When a command or block is obeyed it is first "compiled" into NewBrain BASIC "object code" (this compilation includes the setting up of appropriate datastructures) and then the "object code" is "executed". When a program is being run the object code is kept and so once obeyed a command can be obeyed again without having to be compiled again. This mode of operation saves time and optimises the performance of programs.

Despite BASIC being a dynamic compiler, all interactive features are present. While an ON BREAK trap is not set the user can normally break into a program by using the STOP key, inspect and alter values and program lines and then CONTINUE execution with all other states preserved.

6 NOTES

(a) Spaces

Spaces are insignificant in almost all places in NewBrain BASIC.

(b) Character Set

NewBrain BASIC allows the user a set of 256 distinct characters. It is assumed that the character set includes the ASCII characters (though it need not include all of them). BASIC distinguishes between upper and lower case alphabetic characters only within string constants and REMark strings. The NewBrain keyboard character set includes all the ASCII characters, viewdata graphics characters and others as well.

(c) Errors

NewBrain BASIC produces over 50 numerically coded error messages to aid debugging.

(d) The depth of nesting of for-blocks, GOSUBs, parentheses etc. is limited only by the total amount of memory available.

(e) Extension

The NewBrain operating system contains an extension mechanism comparable to, but more advanced than simple "trapping". This makes NewBrain "ROM software" real software, not just "firmware". Programs in RAM or ROM can replace or extend programs already present in ROM. In particular additional features can be added to NewBrain BASIC without having to replace the original ROMs. The NewBrain paged memory system enables such extension to be virtually indefinite.

New Brain

Technical Note 15 Issue 1

The NewBrain graphics package

The NewBrain microcomputer has a comprehensive graphics package, which allows the user to draw pictures in high resolution on a TV or monitor display screen. The size of the high resolution area can be selected by the user from BASIC, and the graphics package provides many special features including the ability to choose the scale of this area. The graphics package is implemented as a type of device driver, so that the user can open one or more graphics streams at any time.

Facilities available

The following facilities allow the user to draw lines on the screen, using an imaginary pen.

MOVE (x,y) Moves the pen to (x,y) drawing as it goes, and sets the pen angle to the direction taken.

MOVEBY (d) Moves the pen a distance d in the direction of the pen angle, drawing as it goes.

DRAW (x,y,c) Draws a straight line to the position (x,y) using colour c.

DRAWBY (d,c) Draws a straight line of length d in the direction of the pen angle, using colour c.

The DRAW and DRAWBY commands do not change the pen position or angle.

ARC (d,θ) Draws a series of short straight lines to approximate a circular arc of length d, turning through an angle θ.

PLACE (x,y) Moves the pen to position (x,y) without drawing.

TURN (θ) Turns the pen angle to θ.

TURNBY (θ) Increases the pen angle by θ.

RADIANS Causes angles to be input or output in radians.

DEGREES Causes angles to be input or output in degrees.

BACKGROUND (b) Sets the background colour to b:

0 = off

1 = on

WIPE Sets the entire display area to the background colour.

DOT (x,y,c) Plots a single dot in colour c at position (x,y).

AXES (a,b) Draws axes crossing at the current pen position. Marks the X-axis at spacing of a and the Y-axis at spacing of b. If either parameter is zero, that axis is not marked.

COLOUR (c) Sets the pen colour

The pen colour may be 0-3:

0 No action when drawing

1 Lines are drawn in the colour (black or white) opposite to the background colour

2 Lines are drawn in the same colour as the background

3 Each point on the screen is changed from white to black or from black to white as the line passes through it.

An area of the screen may be marked off by lines or arcs, and then coloured in:

FILL Colours the current position according to the current pen colour, then fills the surrounding area up to an enclosing boundary formed by lines whose resultant colour is the same and by the edges of the screen.

The local screen scale may be changed, and the position of the centre selected:

RANGE (a,b) Sets the coordinate scale so that the width of the screen is a and its height is b.

CENTRE (a,b) Sets the origin of the coordinates relative to the bottom left corner of display area.

Text strings may be plotted in the high-resolution area, and the mode in which this is done may be selected:

MODE (m) Sets the mode parameter (only relevant to plotting text strings at present).

Using the graphics from BASIC

The graphics commands listed above are used by entering PLOT followed by one or more of the items in the list, separated by commas.

E.g. PLOT PLACE (16,6), MOVE (16,46), MOVE (80,46), MOVE (16,46)

The PLOT statement may contain a string or a number, which will be plotted starting at the pen position.

E.g. the result R of a computation might be shown at the bottom of the screen with PLOT RANGE (110,100), PLACE (10,0), "RESULT=", PLACE (38,0), R

Another facility – the PEN function – can be used in a BASIC print or assignment statement. It allows the user to ascertain the location of the pen on the screen, and certain other items.

E.g. PRINT PEN (0), PEN (1): REM position

PW = PEN (9)

The values which can be accessed via the PEN function are

0 X – coordinate of pen

1 Y – coordinate of pen

- 2 pen angle
- 3 pen colour
- 4 background colour
- 5 mode
- 6 colour of point at current pen position
- 7 address of first memory location in the high-resolution display
- 8 address of last memory location in the high-resolution display, plus 1
- 9 width of the display (in pixels)

A graphics stream is set up using OPEN to open a stream of type 11. This will normally share the memory area of the console stream, stream zero. As a high resolution display occupies more memory than a character screen display, the shared area needs to be rather large.

e.g. OPEN # 0,0, "170"

OPEN # 1,11

A graphics stream need not share the memory of the console stream if another stream is available with a sufficiently large screen area.

e.g. OPEN # 6,0,1, "170"

OPEN # 1,11, "#6"

The height of the graphics area required is given in the OPEN statement. For 200 graphics rows, which occupy the same display area in the TV or monitor as 20 rows of characters, OPEN # 1,11, "200" or OPEN # 1,11, "#6,200" is used. The OPEN statement may also be used to select the width.

The graphics area may occupy the full width of the screen or a narrower part thereof. The selection is made by adding "W" or "N" in the parameter string before the height.

e.g. OPEN # 1,11, "W200" or OPEN # 1,11, "#6, W200" for a wide screen,
OPEN # 1,11 "N200" or OPEN # 1,11, "#6, N200" for a narrower screen.

The Operating System remembers the first graphics stream opened, and this is used by BASIC for each PLOT statement or PEN function. If more than one graphics stream is opened, PLOT statements may be directed to one or the other by adding the stream number, as for PUT or PRINT:

PLOT # 1, "GRAPHICS STREAM 1"

The pen function may also be used with a stream number to get information from each graphics stream, as for example

PRINT PEN (# 3, 0), PEN (# 3, 1)

to print the X and Y coordinates of the pen position belonging to stream 3.

Technical details

1 OPEN

The parameter string gives the *linked stream*, the *width option*, and the *height* selected for the graphics stream.

Linked stream is "# integer" or null and defaults to 0, i.e. the console device, and determines the stream whose display area is to be shared. The selected stream must be a screen device, whose height is sufficient to accommodate the requested height. Graphics lines require up to ten times as much memory as the character lines they replace.

Width option is "W" or "N" (default is "W"), and determines whether the full width of the screen or a narrower part of it is to be used. Selecting "N" reduces the memory requirement by 20%.

Height is "integer" (or "", integer" if the width option is omitted), and determines the number of graphics lines on the page. The default is 150. The maximum height is 229 if a wide screen is selected and 230 for a narrower screen.

2 Memory Required

The linked stream must be a screen-type stream, opened in advance and of a large enough size to accommodate the desired graphics area. To provide a graphics stream with width option "w" and height 150-159, an original screen size of 110 is required. An original screen size of 114 would provide some extra memory to enable the FILL command to be used more effectively. For each extra 10 lines of graphics height, the original screen size should be increased by 6 lines.

When the width option is "n", an original screen size of 86 is required to provide a graphics stream with height 141-150. An original screen size of 90 would provide some extra memory to enable the FILL command to be used more effectively. For each extra 10 lines of graphics height, the original screen size should be increased by 5 lines.

3 Initial Values

When a graphics stream is opened, the following initial values are set up:

X-Range 1
Y-Range 1

Origin	Bottom left-hand corner
Pen position	(0,0)
Pen angle	0
Angles in	Radians
Background	0
Pen colour	1
Mode	0

4 Text Plotting

Either a string or an expression may appear in a PLOT statement; an expression is first converted to a string, as by STR\$, and will always appear with a leading and trailing space. Each character is plotted in turn, in a matrix of 8 x 10 pixels, with the lower left corner of the first character at the current pen position. Each of the 80 pixels in the matrix is coloured, using the pattern of the character as a chart for this operation. If the point in the character pattern is part of the surround, the corresponding point of the screen is plotted using the MODE value 0, 1, 2 or 3. If the point in the character pattern is part of the character itself, the corresponding point on the screen is plotted using the COLOUR value 0, 1, 2 or 3. So the statement

```
PLOT COLOUR (2), MODE (1), "NewBrain"
```

produces the text "NewBrain" on a reversed-field, and

```
PLOT COLOUR (1), MODE (2), "NewBrain"
```

produces the text "NewBrain" in normal colouring, and clears blank space around the text. A Page zero location is used to hold the address of the 256 character charts, to allow user definable characters in the high resolution area.

5 Default Graphics Stream Number

Each time a graphics stream is opened, its own stream number is inserted into the Page Zero location "default graphics stream number" if and only if that location contained the value zero. Each time a graphics stream is closed, the contents of the "default graphics stream number" location are set to zero if and only if that location contained the number of the closing stream. This ensures that if a user only has one graphics stream open at a time, the PLOT and PEN statements will always refer to that stream.

6 Graphics Control Strings

The PLOT and PEN commands enable a graphics stream to be controlled directly from BASIC in a straight-forward fashion. Each PLOT item, other than a text string, is coded by BASIC as a sequence of from two to twenty bytes and output to the driver, which then interprets this sequence and carries out the instruction. The command sequence is as follows:

First byte:	value 16
Second byte:	plot command number
Bytes 3 to 8:	first parameter, if required
Bytes 9 to 14:	second parameter, if required
Bytes 15 to 20:	third parameter, if required.

Each parameter is a six-byte floating point number, output low byte first. The command numbers, and expected number of parameters, are as follows:

0 MOVE (2)	10 RANGE (2)
1 TURN (1)	11 CENTRE (2)
2 ARC (2)	12 MOVEBY (1)
3 TURNBY (1)	12 DRAW (3)
4 MODE (1)	14 DRAWBY (2)
5 FILL (none)	15 BACKGROUND (1)
6 COLOUR (1)	16 WIPE (none)
7 reserved for text strings	17 AXES (2)
8 used to output PEN	18 PLACE (2)
parameter (1)	19 RADIANS (none)
9 DOT (3)	20 DEGREES (none)

For a text string, the command sequence is:

First byte:	16
Second byte:	7
Third byte:	number of characters n
Fourth byte:	0
Bytes 5 to 4 + n:	text

After command 8 is output with its parameter, six bytes may be input from the stream which will make up the floating-point value of the selected PEN function.

All these control sequences are transparent to the user, as BASIC always constructs the necessary sequences from the PLOT commands.

New Brain

Technical Note 16 Issue 1

Screen Editor - Technical Specification

General Description

The NewBrain keyboard-Screen-vf display editor, XIO, is an interactive input/output device for communication between a user or operator of the NewBrain and a NewBrain user-program such as BASIC. The editor interfaces with the NewBrain Input/Output System, IOS.

Communication between the editor and the program is via the five standard IOS commands, OPENIN, OPENOUT, INPUT, OUTPUT and CLOSE. Since the editor is an Input/Output device OPENIN is equivalent to OPENOUT.

The Displays

The TV/video display editor holds a page of between 1 and 255 lines of 40 or 80 characters per line. The screen display will show 24 or 30 lines and this window is scrolled up and down the page. IOS allows for multiple copies of a device to be open; thus, memory allowing, up to 255 pages can be simultaneously maintained.

The vf display editor holds a single line between 16 and 254 characters in length. The 16 character vf display window is scrolled backwards and forwards over the line. The editor waits for the user to press the NEWLINE key before displaying a new line. This wait can be suppressed by outputting the appropriate control codes. During the wait the window can be scrolled from side to side by the cursor control keys.

The vf display can also be used as a window on a screen page, displaying the part of the current line around the cursor.

Character Sets

The vf display character set consists of the 64 ASCII upper case characters, excepting character 95 for which the £ sign is substituted, and 64 graphics characters. All characters can be blinked. The vf editor uses only 65 of these characters and reserves blinking as an indication of cursor position. Characters sent to the editor are recorded before display, so that lower case letters are displayed as their upper case equivalents, and so the coding agrees with that used for the screen display.

The screen display character set depends on the character generator ROM fitted. 512 characters are available, though at most 255 characters can be displayed at a time. Character set selection is achieved by the "Set TV Mode" (control W) code. In certain modes 127 characters are available for display, in normal or reverse field. According to mode the background can also be set to forward or reverse field. Although 32 codes are used for controls, all 255 displayable characters can be used by using the ESCAPE control code. The code 0 is used by the hardware for control and cannot be displayed.

Open

Parameters supplied on open are as follows:

1 Device Number

- 0 TVIO - video screen editor only
- 3 LIIO "vf single line editor ("line image")
- 4 TLIO - video screen editor, using vf display as a window on the screen image

2 Port Number

No use is made of this by the device. Thus this can be used to make IOS open multiple copies of the device.

3 Parameter String

For TVIO & TLIO this consists of two optional fields:

- length - A single character "s", "S", "l" or "L". S indicates forty character lines, L indicates 80 character lines. Default is 40.
- depth - An integer in the range 1..255, being the number of 40 or 80 character lines on the page. Default is 24.

For LIIO this is a single optional field, an integer in the range 16..255 being the number of characters in the "line image".

Note that TVIO and TLIO behave the same way as LIIO when the requested depth is 1.

Input

Under normal circumstances INPUT causes a line of characters to be collected from the keyboard interaction and the first character of the line to be returned. Subsequent INPUT commands will return the rest of the characters of the line in sequence, including the end of line character (NEWLINE, code 13). A further call to INPUT will cause a new line to be collected and returned. Exceptionally a sequence of calls to INPUT will return the character at the cursor position, the line on which the cursor is positioned (without collecting a new line by keyboard interaction), the entire current page or the x-y cursor address.

Keyboard Interaction

The keyboard interaction is started when a call is made to INPUT when there are not already characters set to be returned by the editor. The interaction continues until the NEWLINE key is pressed. The cursor is always displayed during the interaction. If the interaction starts with the cursor at the start of a line all control codes are available through the keyboard and have the same effect as if OUTPUT. If the interaction starts with the cursor within a line, backspacing past that position and codes which move the cursor out of the line on which it is resting are suppressed. The part of the line up to the cursor position when the interaction was started (the "prompt") is not returned by INPUT. The special control keys on the keyboard and the main keys used with the CONTROL key generate control codes. Normally the main keys generate lower case letters, in conjunction with SHIFT upper case letters, in conjunction with GRAPHICS codes in the range 128-255 ("graphics codes"). Keys 0..9 in conjunction with CONTROL change the "keyboard mode" - i.e. the correspondence between the key pressed and the code generated; but these keys do not themselves produce a character code. Control and 1 enters "SHIFT Lock", CONTROL and 0 returns the keyboard mode to normal.

Output

Calls to output cause a character to be put onto the screen or vt display. Thirty-two characters are used as control codes, accessing special features of the editor. The effect of control codes other than NEWLINE is the same whether entered via OUTPUT or entered via the keyboard interaction in response to a call to INPUT, when the cursor is at the beginning of a line.

Screen Lines, Cursor Display, Autodelete Feature

Lines on the screen may be of any length up to the size of page. Lines longer than 40 or 80 characters wrap round, a continuation character being displayed at the start of each screen line other than the first to indicate that this is a continuation line. The cursor is displayed as a blinking underline or a blinking block. The latter mode of display is used for two purposes. When the cursor is beyond the right end of the line - i.e. the next character entered will be the first character of a subsequent continuation, it is displayed at the right-most position in the line as a block. This contrasts with being displayed in the right-most position as an underline when the next character entered is to be entered at that position. The other use of the block cursor is at the left-most position of a line just after a NEWLINE has been entered. This indicates that if the next character to be entered is not a control code the present contents of the line will be cleared to spaces. This "autodelete" feature enables a page to be overprinted without having to clear outdated information. The autodelete mode is cancelled if the first character entered is a control code, so that overprinting of forms etc. where information previously printed is to be retained, can be achieved.

Use as BASIC Console

BASIC opens an editor device on Stream 0 when started up. The parameters used for this depend on the hardware. For a NewBrain D, MDB LIIO with a length of 80 is used, for A or AD TVIO with a length of 40 and depth of 24, for M, MD TVIO with a length of 80 and depth 24. Port number 0 is used.

Control Codes

Hex	Decimal	Control	Key	
0	0	@		Null
1	1	A	Sh/insert	Insert line
2	2	B	Sh/l	
3	3	C	Cntl/newline	Send page
4	4	D		End of file
5	5	E		Send line
6	6	F		Show cursor
7	7	G		Cursor off
8	8	H	--	Cursor left
9	9	I	Cntl/escape	Tab 8 spaces
A	10	J	↓	Cursor down
B	11	K	↑	Cursor up
C	12	L	Home	Cursor home
D	13	M	Newline	Newline
E	14	N	Sh/l	Attribute on
F	15	O	Sh/escape	Attribute off
10	16	P		Graphics escape
11	17	Q	Insert	Enter insert mode
12	18	R	Grph/l	Make new line
13	19	S	Grph/l	Make continuation line
14	20	T	Grph/newline	Send cursor character
15	21	U	Grph/insert	Send x, y
16	22	V	Cntl/insert	Set cursor x, y
17	23	W	Grph/escape	Set TV control

Screen Editor - Technical Specification

General Description

The NewBrain keyboard-Screen-vf display editor, XIO, is an interactive input/output device for communication between a user or operator of the NewBrain and a NewBrain user-program such as BASIC. The editor interfaces with the NewBrain Input/Output System, IOS.

Communication between the editor and the program is via the five standard IOS commands, OPENIN, OPENOUT, INPUT, OUTPUT and CLOSE. Since the editor is an Input/Output device OPENIN is equivalent to OPENOUT.

The Displays

The TV/video display editor holds a page of between 1 and 255 lines of 40 or 80 characters per line. The screen display will show 24 or 30 lines and this window is scrolled up and down the page. IOS allows for multiple copies of a device to be open; thus, memory allowing, up to 255 pages can be simultaneously maintained.

The vf display editor holds a single line between 16 and 254 characters in length. The 16 character vf display window is scrolled backwards and forwards over the line. The editor waits for the user to press the NEWLINE key before displaying a new line. This wait can be suppressed by outputting the appropriate control codes. During the wait the window can be scrolled from side to side by the cursor control keys.

The vf display can also be used as a window on a screen page, displaying the part of the current line around the cursor.

Character Sets

The vf display character set consists of the 64 ASCII upper case characters, excepting character 95 for which the £ sign is substituted, and 64 graphics characters. All characters can be blinked. The vf editor uses only 65 of these characters and reserves blinking as an indication of cursor position. Characters sent to the editor are recorded before display, so that lower case letters are displayed as their upper case equivalents, and so the coding agrees with that used for the screen display.

The screen display character set depends on the character generator ROM fitted. 512 characters are available, though at most 255 characters can be displayed at a time. Character set selection is achieved by the "Set TV Mode" (control W) code. In certain modes 127 characters are available for display, in normal or reverse field. According to mode the background can also be set to forward or reverse field. Although 32 codes are used for controls, all 255 displayable characters can be used by using the ESCAPE control code. The code 0 is used by the hardware for control and cannot be displayed.

Open

Parameters supplied on open are as follows:

1 Device Number

- 0 TVIO - video screen editor only
- 3 LIIO "vf single line editor ("line image")
- 4 TLIO - video screen editor, using vf display as a window on the screen image

2 Port Number

No use is made of this by the device. Thus this can be used to make IOS open multiple copies of the device.

3 Parameter String

For TVIO & TLIO this consists of two optional fields:

- length - A single character "s", "S", "l" or "L". S indicates forty character lines, L indicates 80 character lines. Default is 40.
- depth - An integer in the range 1..255, being the number of 40 or 80 character lines on the page. Default is 24.

For LIIO this is a single optional field, an integer in the range 16..255 being the number of characters in the "line image".

Note that TVIO and TLIO behave the same way as LIIO when the requested depth is 1.

Input

Under normal circumstances INPUT causes a line of characters to be collected from the keyboard interaction and the first character of the line to be returned. Subsequent INPUT commands will return the rest of the characters of the line in sequence, including the end of line character (NEWLINE, code 13). A further call to INPUT will cause a new line to be collected and returned. Exceptionally a sequence of calls to INPUT will return the character at the cursor position, the line on which the cursor is positioned (without collecting a new line by keyboard interaction), the entire current page or the x-y cursor address.

Keyboard Interaction

The keyboard interaction is started when a call is made to INPUT when there are not already characters set to be returned by the editor. The interaction continues until the NEWLINE key is pressed. The cursor is always displayed during the interaction. If the interaction starts with the cursor at the start of a line all control codes are available through the keyboard and have the same effect as if OUTPUT. If the interaction starts with the cursor within a line, backspacing past that position and codes which move the cursor out of the line on which it is resting are suppressed. The part of the line up to the cursor position when the interaction was started (the "prompt") is not returned by INPUT. The special control keys on the keyboard and the main keys used with the CONTROL key generate control codes. Normally the main keys generate lower case letters, in conjunction with SHIFT upper case letters, in conjunction with GRAPHICS codes in the range 128-255 ("graphics codes"). Keys 0..9 in conjunction with CONTROL change the "keyboard mode" - i.e. the correspondence between the key pressed and the code generated; but these keys do not themselves produce a character code. Control and I enters "SHIFT Lock", CONTROL and O returns the keyboard mode to normal.

Output

Calls to output cause a character to be put onto the screen or vt display. Thirty-two characters are used as control codes, accessing special features of the editor. The effect of control codes other than NEWLINE is the same whether entered via OUTPUT or entered via the keyboard interaction in response to a call to INPUT, when the cursor is at the beginning of a line.

Screen Lines, Cursor Display, Autodelete Feature

Lines on the screen may be of any length up to the size of page. Lines longer than 40 or 80 characters wrap round, a continuation character being displayed at the start of each screen line other than the first to indicate that this is a continuation line. The cursor is displayed as a blinking underline or a blinking block. The latter mode of display is used for two purposes. When the cursor is beyond the right end of the line - i.e. the next character entered will be the first character of a subsequent continuation, it is displayed at the right-most position in the line as a block. This contrasts with being displayed in the right-most position as an underline when the next character entered is to be entered at that position. The other use of the block cursor is at the left-most position of a line just after a NEWLINE has been entered. This indicates that if the next character to be entered is not a control code the present contents of the line will be cleared to spaces. This "autodelete" feature enables a page to be overprinted without having to clear outdated information. The autodelete mode is cancelled if the first character entered is a control code, so that overprinting of forms etc, where information previously printed is to be retained, can be achieved.

Use as BASIC Console

BASIC opens an editor device on Stream 0 when started up. The parameters used for this depend on the hardware. For a NewBrain D, MDB LIIO with a length of 80 is used, for A or AD TVIO with a length of 40 and depth of 24, for M, MD TVIO with a length of 80 and depth 24. Port number 0 is used.

Control Codes

Hex	Decimal	Control	Key	
0	0	@		Null
1	1	A	Sh/insert	Insert line
2	2	B	Sh/I	
3	3	C	Cntl/newline	Send page
4	4	D		End of file
5	5	E		Send line
6	6	F		Show cursor
7	7	G		Cursor off
8	8	H	-	Cursor left
9	9	I	Cntl/escape	Tab 8 spaces
A	10	J	↓	Cursor down
B	11	K	↑	Cursor up
C	12	L	Home	Cursor home
D	13	M	Newline	Newline
E	14	N	Sh/I	Attribute on
F	15	O	Sh/escape	Attribute off
10	16	P		Graphics escape
11	17	Q	Insert	Enter insert mode
12	18	R	Grph/I	Make new line
13	19	S	Grph/I	Make continuation line
14	20	T	Grph/newline	Send cursor character
15	21	U	Grph/insert	Send x, y
16	22	V	Cntl/insert	Set cursor x, y
17	23	W	Grph/escape	Set TV control

18	24	X	Sh/←	Delete left
19	25	Y	Sh/→	Delete character
1A	26	Z	→	Cursor right
1B	27	[Escape	Escape next character
1C	28	/	Cntl/←	Cursor home left
1D	29]	Cntl/→	Cursor home right
1E	30	!	Cntl/home	Clear line
1F	31		Shift/home	Clear page

Interpretation of Control Codes

- 0 No action.
- 1 A line of spaces is inserted at the cursor, the cursor line and subsequent lines are shifted down. The last line is lost.
- 2 The whole cursor line including the lines of which it is a continuation (if any) and any continuations of it is deleted. Subsequent lines are shifted up, sufficient lines of spaces being inserted at the end.
- 3 Subsequent calls to INPUT will return all the characters on the page, including NEWLINES. This mode is cancelled when all characters are returned, or by a call to OUTPUT. After returning all characters the cursor will be in the top left "home position" on the screen.
- 4 No action (this code has a special meaning in BASIC).
- 5 Subsequent calls to INPUT will return all the characters of the current line, including the NEWLINE. Cancelled as code 3.
- 6 The cursor will be displayed at all times until code 7 is entered.
- 7 The cursor will not be displayed, except during keyboard interaction, until code 6 is entered.
- 8 Move cursor left one space. This is not possible if at the top left-most position on a line.
- 9 Move cursor right at least one space and sufficiently many spaces to bring it to a screen column a multiple of 8 spaces from the start, or failing that, to the start of the next line or to the space after the continuation character on the next line.
- 10 Move cursor vertically down one space. This is not possible if on the bottom line of the screen.
- 11 Move cursor vertically up one space. This is not possible if on the top line of the screen.
- 12 Home cursor to top left of screen.
- 13 Move cursor to the start of the next line. In INPUT end keyboard interaction. On the bottom line scroll screen up losing top line.
- 14 Invert the top bit of all subsequent non-control codes before displaying. Subsequent characters 32...127 will become 160...255, 129...255 will become display codes 1...127. (128 will become 0 which is not displayable and is always treated as a NULL control code). This mode is cancelled only when code 15 is entered.
- 15 Cancel mode introduced by code 14.
- 16 No action - this code is reserved for use by a high resolution display editor.
- 17 Subsequent characters will be inserted at the cursor position, characters to the right on the line being scrolled right and, when appropriate, down (causing subsequent lines to be scrolled down and the bottom line lost). If the line already fills the "line image" (LIO) or the screen page (TVIO or TLIO) this is not possible. This mode is cancelled when any control code is entered.
- 18 The current screen line, if it is a continuation line, is made the start of a new line. Continuations of it are scrolled back and up if required.
- 19 The current screen line is made a continuation of the line above it. Subsequent continuations of it are scrolled down and right if necessary.
- 20 A subsequent call to INPUT will return the character at the cursor position. This mode is cancelled in the same way as code 3.
- 21 Two subsequent calls to INPUT will return the "x-y cursor address", firstly the x address which is the horizontal displacement of the cursor from the left of the screen starting at 1, then the y address which is the vertical displacement from the top of the screen, starting at 1 (irrelevant in the case of LIO). Cancelled as code 3.
- 22 The two bytes next entered are interpreted as an x-y cursor address (see code 21), and the cursor is set to the given position. This mode is cancelled when the two bytes have been received by the editor, or if a call to INPUT intervenes.
- 23 Set TV control, certain bits of the next byte entered are loaded into the video hardware "TV control register". The bits are currently used as follows:
bit
3 1 = 8 lines/character, upper Character ROM set
0 = 10 lines/character, lower Character ROM set
1 1 = full character set

0 = half character set, top bit used to reverse character field
0 1 = black background
0 = white

This mode is cancelled in the same way as code 22.

- 24 Delete the character to the left of the cursor moving the cursor left one space. This has no effect if at the start of a line (not a continuation line) or if the character is outside the LIO window. The rest of the line to the right of the cursor is scrolled to the left, and it and subsequent lines up if necessary.
- 25 Delete the character at the cursor. Otherwise as code 24.
- 26 Move cursor right one space. this is not possible if at the end of a line.
- 27 Put the next character directly into the display – i.e. do not treat it as a control code. This mode is cancelled after the next character has been entered, or if a call to INPUT intervenes.
- 28 Send cursor to the left-most position on this line.
- 29 Send cursor to the right-most position on this line.
- 30 Replace the current line by a line of spaces, scrolling up subsequent lines if appropriate. Send cursor to beginning of line.
- 31 Clear the whole screen to spaces and send cursor to “home” top left.